

ESTRUCTURES DE DADES I ALGORISMES

COL·LECCIÓ D'EXÀMENS

Albert Atserias

Amalia Duch

Albert Oliveras

Enric Rodríguez Carbonell

(Editors)

7 de febrer de 2022



Departament de Ciències de la Computació
Universitat Politècnica de Catalunya

Taula de continguts

1	Exàmens Parcial	1
2	Exàmens d'Ordinador	117
3	Exàmens Finals	125
4	Solucions d'Exàmens Parcial	279
5	Solucions d'Exàmens d'Ordinador	345
6	Solucions d'Exàmens Finals	417

Exàmens Parciais

Examen Parcial EDA - torn 1

Duració: 2 hores

18/10/2010

Problema 1

(1 punt)

- Doneu la definició de $O(f)$:

- El teorema mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma $T(n) = aT(n/b) + \Theta(n^k)$ amb $b > 1$ i $k \geq 0$, llavors, fent $\alpha = \log_b a$,

$$T(n) = \begin{cases} \text{ } & \text{si } \alpha < k, \\ \text{ } & \text{si } \alpha = k, \\ \text{ } & \text{si } \alpha > k. \end{cases}$$

Problema 2

(2 punts)

Ompliu els blancs de la forma més precisa possible.

- Quicksort ordena n elements en temps en el cas pitjor.
- Per multiplicar dues matrius grans eficientment podem fer servir l'algorisme de .
- $2 + \cos(n) = \Theta(\text{ })$.
- $2\sqrt{n} + 1 + n + n^2 = \Theta(\text{ })$.
- $\log n + \log \log(n^2) = \Theta(\text{ })$.
- $\frac{n^2 - 6n}{2} + 5n = \Theta(\text{ })$.
- $n^2 - 3n - 18 = \text{ } (n)$.
- Si

$$T(n) = \begin{cases} 4 & \text{si } n = 1, \\ 3T(n/2) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores, $T(n) = \Theta(\text{ })$.

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 2T(n-1) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores, $T(n) = \Theta(\text{ })$.

Problema 3**(1.5 punts)**

Sigui $c \in \mathbb{R}$ i sigui h una funció de \mathbb{N} a \mathbb{R} . Recordeu que

$$\lim_{n \rightarrow \infty} h(n) = c \iff \forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : |h(n) - c| \leq \epsilon.$$

Sigui $c \in \mathbb{R}$ i siguin f i g dues funcions de \mathbb{N} a \mathbb{R} . Demostreu que

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \implies f \in O(g).$$

Problema 4**(1 punt)**

Suposeu que $\text{primer}(n)$ és una crida a una funció amb temps $O(\sqrt{n})$.

Considereu un procediment amb cos principal:

```
if (primer(n)) A;  
else B;
```

Doneu fites senzilles i ajustades amb notació O per al temps d'aquest procediment, en funció de n , suposant que:

1. A triga $O(n)$ i B triga $O(1)$.
2. A i B , ambdòs, triguen $O(1)$.

Problema 5**(1 punt)**

```
int misteri (int n) {  
    if (n == 1) return 1;  
    return misteri (n-1) + 2*n - 1;  
}
```

- Diguen què calcula la funció *misteri*.
- Doneu el seu cost en funció de n .

Problema 6**(2 punts)**

Dissenyau un algorisme de temps $O(n \log m)$ que trobi la unió $A \cup B$ de un conjunt A de n elements amb un conjunt B de m elements, amb $m \leq n$. Els conjunts A i B son de nombres reals i estan representats per vectors no necessàriament ordenats i sense elements repetits. La sortida és un vector, no necessàriament ordenat, sense elements repetits que representa la unió.

Quins canvis requereix el vostre algorisme si es demana la intersecció en comptes de la unió?

Problema 7

(1.5 punts)

```

void misteri_2 (vector<int>& T, int e, int m1, int m2, int d) {
    vector<int> B(d-e+1);
    int i = e, j = m1+1, k = m2+1, l = 0;
    while (i ≤ m1 and j ≤ m2 and k ≤ d) {
        if (T[i] ≤ T[j] and T[i] ≤ T[k]) B[l++] = T[i++];
        else if (T[j] ≤ T[k]) B[l++] = T[j++];
        else B[l++] = T[k++];
    }
    while (i ≤ m1 and j ≤ m2) {
        if (T[i] ≤ T[j]) B[l++] = T[i++];
        else B[l++] = T[j++];
    }
    while (i ≤ m1 and k ≤ d) {
        if (T[i] ≤ T[k]) B[l++] = T[i++];
        else B[l++] = T[k++];
    }
    while (j ≤ m2 and k ≤ d) {
        if (T[j] ≤ T[k]) B[l++] = T[j++];
        else B[l++] = T[k++];
    }
    while (i ≤ m1) B[l++] = T[i++]; while (j ≤ m2) B[l++] = T[j++];
    while (k ≤ d) B[l++] = T[k++];
    for (l=0; l ≤ d-e; ++l) T[e+l] = B[l];
}

void misteri_1 (vector<int>& T, int e, int d) {
    if (e+1 == d) {
        if (T[e] > T[d]) swap(T[e], T[d]);
    }
    if (e+1 < d) {
        int m1 = e + (d-e+1)/3;
        int m2 = e + 2*(d-e+1)/3;
        misteri_1 (T, e, m1); misteri_1 (T, m1+1, m2); misteri_1 (T, m2+1, d);
        misteri_2 (T, e, m1, m2, d);
    }
}

void misteri (vector<int>& T) {
    misteri_1 (T, 0, T.size ()-1);
}

```

De quin algorisme clàssic és variant aquest algorisme? Quin és el seu cost?

Examen Parcial EDA - torn 2

Duració: 2 hores

18/10/2010

Problema 1

(1 punt)

- Doneu la definició de $\Omega(f)$:

- El teoreme mestre de resolució de recurrències substractores afirma que si tenim una recurrència de la forma $T(n) = aT(n - c) + \Theta(n^k)$ amb $c > 0$ i $k \geq 0$, llavors,

$$T(n) = \begin{cases} \text{[]} & \text{si } a < 1, \\ \text{[]} & \text{si } a = 1, \\ \text{[]} & \text{si } a > 1. \end{cases}$$

Problema 2

(2 punts)

Ompliu els blancs de la forma més precisa possible.

- Per multiplicar dos naturals molts llargs eficientment podem fer servir l'algorisme de .
- Per ordenar n elements en temps $\Theta(n \log n)$ en el cas pitjor podem servir l'algorisme de .
- Multipliqueu $\log n + 6 + O(1/n)$ per $n + O(\sqrt{n})$. El resultat simplificat tan com possible és $O(\text{[]})$.
- $\sin(n) + 10 + n = \Theta(\text{[]})$.
- $\frac{1}{3}n^2 + 3n \log n + 5n^8 = \Theta(\text{[]})$.
- $n\sqrt{n} + n^2 = \Theta(\text{[]})$.
- $3n \log n = \text{[]}(n^2)$.
- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 3T(n/2) + n - 2 & \text{si } n > 1. \end{cases}$$

aleshores, $T(n) = \Theta(\text{[]})$.

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 4T(n-1) + n & \text{si } n > 1. \end{cases}$$

aleshores, $T(n) = \Theta(\text{[]})$.

Problema 3**(1.5 punts)**

Sigui $c \in \mathbb{R}$ i sigui h una funció de \mathbb{N} a \mathbb{R} . Recordeu que

$$\lim_{n \rightarrow \infty} h(n) = c > 0 \iff \forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : |h(n) - c| \leq \epsilon.$$

Sigui $c \in \mathbb{R}$ tal que $c > 0$ i siguin f i g dues funcions de \mathbb{N} a \mathbb{R} . Demostreu que

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c \implies f \in \Omega(g).$$

Problema 4**(1 punt)**

```

int misteri (int m, int n) {
    int result = 0;
    while (m > 0) {
        if (m % 2 != 0) result += n;
        m /= 2; n *= 2;
    }
    return result ;
}

```

- Diguen què calcula la funció *misteri*.

- Doneu el seu cost en funció de m .

Problema 5**(1 punt)**

Considereu un procediment amb cos principal:

```

int k = f(n);
int s = 0;
for (int i = 1; i ≤ k; ++i) s += i;

```

amb $f(n)$ una crida a la funció f .

Doneu fites senzilles i ajustades amb notació O per al temps d'aquest procediment, en funció de n , suposant que:

1. El temps de $f(n)$ és $O(n)$ i el valor de $f(n)$ és $n!$.

2. El temps de $f(n)$ és $O(n)$ i el valor de $f(n)$ és n .

3. El temps de $f(n)$ és $O(n^2)$ i el valor de $f(n)$ és $n!$.

4. El temps de $f(n)$ és $O(1)$ i el valor de $f(n)$ és 0.

Problema 6**(2.5 punts)**

Donada una seqüència A de n enters representada per un vector no necessàriament ordenat, es vol un algorisme de temps $O(n \log n)$ per determinar si A conté més de $n/2$ elements iguals.

1. Ordenant el vector amb un algorisme d'ordenació de temps $O(n \log n)$ és immediat resoldre aquest problema. Digueu com.
2. Supposeu que els elements només es poden comparar per igualtat ($=$, \neq) però no es poden comparar per ordre ($<$, \leq , $>$, \geq). Proposeu un algorisme de Dividir i Vèncer de temps $O(n \log n)$ que resolgui el problema.

Problema 7**(1 punt)**

```
int misteri_2 (vector<int>& T, int e, int d) {
    int x = T[d];
    int i = e-1;
    for (int j = e; j <= d; ++j) {
        if (T[j] <= x) swap(T[++i], T[j]);
    }
    if (i < d) return i;
    return i-1;
}
```

```
void misteri_1 (vector<int>& T, int e, int d) {
    if (e < d) {
        int q = misteri_2 (T, e, d);
        misteri_1 (T, e, q);
        misteri_1 (T, q+1, d);
    }
}
```

```
void misteri (vector<int>& T) {
    misteri_1 (T, 0, T.size ()-1);
}
```

De quin algorisme clàssic es tracta? Quin és el seu cost quan tots els elements són iguals?

Examen Parcial EDA

Duració: 2 hores

21/03/2011

Problema 1

(5 punts)

Ompliu els blancs de la forma més precisa possible.

- Insertion sort ordena n elements en temps en el cas millor.
- Trobar la mediana de n elements té una complexitat en temps $\Omega(\text{ })$.
- Com de ràpid es poden multiplicar una matriu $kn \times n$ per una $n \times kn$ fent servir l'algorisme de Strassen? .
- Quantes línies, en funció de n i notació Θ , imprimeix el següent programa? Escriviu una recurrència i resoleu-la.

```
//pre: n ≥ 1
void f(int n) {
    if (n > 1) {
        cout << "segueix" << endl;
        f(n/2);
        f(n/2);
    }
}
```

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n < 3, \\ 9T(n/3) + n^2 & \text{si } n \geq 3. \end{cases}$$

Aleshores, $T(n) = \Theta(\text{ })$.

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n = 1, \\ 3T(n-1) + n^3 & \text{si } n > 1. \end{cases}$$

Aleshores, $T(n) = \Theta(\text{ })$.

- Multipliqueu 101×101 (en binari) fent servir l'algorisme de Karatsuba. Mostreu els passos que heu seguit.
- Les taules següents representen etapes de l'execució dels algorismes d'ordenació per fusió, ràpida, per inserció i per selecció (mergesort, quicksort, insertion sort i selection sort) aplicats a la taula:

10	2	5	3	7	13	1	6
----	---	---	---	---	----	---	---

 Escriviu al costat de cada taula a quin algorisme dels anteriors correspon (suposeu que hi ha una taula de cada algorisme).

2	3	5	10	1	6	7	13	
2	3	5	10	7	13	1	6	
1	2	5	3	7	6	13	10	
1	2	3	5	6	13	10	7	

- Indiqueu per a cada parell de funcions (A, B) a la taula següent si A és O , Ω o Θ de B . Supposeu que $k \geq 1$, $\epsilon > 0$ i $c > 1$ són constants. La resposta ha de ser “sí” o “no” a cada casella buida.

A	B	O	Ω	Θ
$\log^k(n)$	n^ϵ			
$\log n$	$\log \log(n^2)$			
n^k	c^n			
2^{n+1}	2^n			
2^{2n}	2^n			

Problema 2

(2.5 punts)

```

int misteri_2 (vector<int>& T, int e, int d) {
    int x = T[d];
    int i = e-1;
    for (int j = e; j ≤ d; ++j) {
        if (T[j] ≤ x) swap(T[++i], T[j]);
    }
    if (i < d) return i;
    return i-1;
}

int misteri_1 (vector<int>& T, int e, int d, int k) {
    if (e == d) return T[e];
    int q = misteri_2 (T, e, d);
    if (q - e + 1 ≥ k) return misteri_1 (T, e, q, k);
    return misteri_1 (T, q + 1, d, k - q + e - 1);
}

// pre: 1 ≤ k ≤ n = T.size()
int misteri (vector<int>& T, int k) {
    return misteri_1 (T, 0, T.size()-1, k);
}

int main() {
    int k, x;
    cin >> k;
    vector<int> v;
    while (cin >> x) v.push_back(x);
    cout << misteri(v, k) << endl;
}

```

- Digueu què calcula el programa anterior.
- Doneu una recurrència que descrigui el seu cost en temps en funció de n en el cas pitjor (justifiqueu) i resoleu-la.
- Doneu el seu cost en temps en funció de n en el cas millor (justifiqueu).
- Digueu a quin algorisme clàssic us recorda l'algorisme anterior i perquè.

Problema 3**(2.5 punts)**

Escriuiu un algorisme que, donat un natural $N \geq 0$, calculi $\lfloor \sqrt{N} \rfloor$ (la part entera per defecte de l'arrel quadrada de N) en temps $\Theta(\log N)$ en el cas pitjor. Justifiqueu-ne la complexitat.

Ajuda: penseu en la cerca dicotòmica.

Examen Parcial EDA

Duració: 2 hores

20/10/2011

Problema 1

(4,5 punts)

- (0,5 punts) Calculeu $101 + 102 + \dots + 999 + 1000$.
- (0,5 punts) Expliqueu per quina raó l'afirmació "El temps d'execució de l'algorisme A amb entrada n és com a mínim $O(n^2)$ " no significa res.
- (1 punt) Ordeneu les funcions següents segons el seu ordre de creixement asimptòtic: $5\log(n + 100)^{10}$, 2^{2n} , $0.001n^4 + 3n^3 + 1$, $(\ln(n))^2$, \sqrt{n} , 3^n .
- (1,5 punts) Considereu les tres alternatives següents per resoldre un mateix problema:
 - L'algorisme A resol una instància del problema dividint-la en cinc instàncies de la meitat de la talla, resolent recursivament cada instància, i combinant les solucions en temps lineal.
 - L'algorisme B resol una instància de talla n resolent recursivament dues instàncies de talla $n - 1$ i combinant les solucions en temps constant.
 - L'algorisme C resol una instància de talla n dividint-la en nou instàncies de talla $n/3$, resolent recursivament cada instància, i combinant les solucions en temps $\Theta(n^2)$.

Analitzeu el cost de cadascun dels tres algorismes. Quin és el més eficient?

- (1 punt) Sigui x un natural representat en una taula de n bits. Doneu, en funció de n , el cost de calcular $x + 1$ en els casos pitjor i millor.

Problema 2

(3.5 punts)

- (1 punt) Analitzeu el cost temporal de *fusio2* en funció de les mides de a i b :

```
vector<int> fusio2(const vector<int>& a, const vector<int>& b) {
    // Pre: a i b estan ordenats de forma no-decreixent
    int na = a.size ();
    int nb = b.size ();
    vector<int> c(na + nb);
    int ia = 0;
    int ib = 0;
    int j = 0;
    while (ia < na and ib < nb) {
        if (a[ia] <= b[ib]) { c[j] = a[ia]; ++ia; }
        else { c[j] = b[ib]; ++ib; }
        ++j;
    }
    while (ia < na) { c[j] = a[ia]; ++ia; ++j; }
    while (ib < nb) { c[j] = b[ib]; ++ib; ++j; }
    return c;
}
```

- (1 punt) Suposem que tenim k vectors ordenats (de manera no-decreixent), cadascun amb n elements, guardats en un `vector<vector<int>>` de mida k . Volem calcular el `vector<int>` de mida kn resultant de fer la fusió de tots ells.

Un possible algorisme consisteix en fusionar usant *fusio2* el primer i el segon vectors, després fusionar el resultat amb el tercer vector, després amb el quart, etc. Quin és el cost en temps d'aquest algorisme, en funció de k i n ?

- (1,5 punts) Implementeu en C++ una funció

`vector<int> fusio2(const vector<int> &t)`

que resolgui més eficientment el problema. Justifiqueu el seu cost temporal.

Problema 3

(2 punts)

Els nombres de Fibonacci satisfan la recurrència $f_{n+2} = f_{n+1} + f_n$ per a $n \geq 0$, amb $f_0 = 0$ i $f_1 = 1$. Demostreu que, per a $n \geq 1$, se satisfà la identitat següent:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}$$

i aprofiteu-la per donar un algorisme de cost $\Theta(\log n)$ que, donat un natural n , calculi f_n . No es valorarà cap algorisme que no tingui cost $\Theta(\log n)$. No cal escriure codi però cal incloure l'anàlisi de l'algorisme que proposeu.

(1 punt) Demostració:

(1 punt) Descripció i anàlisi de l'algorisme:

Examen Parcial EDA

Duració: 2 hores

19/3/2012

Problema 1

(3,5 punts)

- (1) Per a les funcions f i g següents, digueu si és cert o fals que $f = O(g)$, $f = \Omega(g)$ i $f = \Theta(g)$.

$f(n)$	$g(n)$	$f = O(g)$	$f = \Omega(g)$	$f = \Theta(g)$
\sqrt{n}	$n^{2/3}$			
$\log(2n)$	$\log(3n)$			
$n^{0.1}$	$(\log n)^{10}$			
2^n	2^{n+1}			
$100n + \log n$	$n + (\log n)^2$			

- (0,5) Donat un vector de n elements possiblement repetits, digueu com trobar, en temps $O(n \log n)$, tots els elements repetits.
- (1,5) Volem convertir a binari 10^n (un u seguit de n zeros), on n és una potència estrictament positiva de 2.

Completeu l'algorisme següent:

```
string pwr2bin(int n) {
    if (n == 1) return "1010"; // 10 en binari (8 + 2)
    string z = 
    return Karatsuba(z,z);
}
```

Doneu una recurrència que descrigui el cost temporal de l'algorisme anterior i resoleu-la.

- (0,5) El professor X us diu a classe que és asimptòticament més lent elevar al quadrat un enter de n bits que multiplicar dos enters de n bits. L'heu de creure? Justifiqueu la vostra resposta.

Problema 2

(1,5 punts)

Considereu el codi següent:

```
void f(vector<int>& v, int i, int j) {
    if ((j-i+1)>1) {
        int k = (i+j)/2;
        f(v, i, k);
        if (k%2 == 0) f(v, k+1, j);
        g(v, i, k, j);
    }
}

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
```

```

    for (int i=0; i<n; ++i) cin >> a[i];
    f(a, 0, n-1);
}

```

La funció g no crida a la funció f .

Digueu:

- Com a màxim, quantes crides recursives a la funció f poden haver a la pila de recursió en un moment donat?

a) 2 b) $\Theta(n)$ c) $\Theta(\log n)$ d) $\Theta(1)$

- En total, en el cas pitjor, quantes crides es fan a la funció f ? Escolliu l'opció més precisa.

a) $\Theta(n\sqrt{n})$ b) $O(n)$ c) $O(n^2)$ d) $\Omega(\log n)$

Problema 3

(1,5 punts)

Considereu el codi següent:

// e i d delimiten un interval tancat (potser buit) dins del vector V

```

void misteri (int e, int d, vector<int>& V) {
    if (e < d) {
        barreja (e, d, V); // barreja a l'atzar els elements de V[e..d],
                           // amb cost proporcional a d - e + 1
        misteri (e, d - 1, V);
        int i = e;
        while (V[i] < V[d]) ++i;
        intercanvia (V[i], V[d]); // cost constant
        misteri (i + 1, d, V);
    }
}

```

Digueu, justificadament, què fa l'acció `misteri`, i quin cost té en el cas pitjor i millor.

Problema 4

(1 punt)

Siguin $f, g : \mathbb{N} \rightarrow \mathbb{R}^{\geq 1}$ dues funcions tals que $\lim_{n \rightarrow \infty} g(n) = \infty$, on $\mathbb{R}^{\geq 1}$ representa el conjunt dels nombres reals més grans o iguals que 1.

- És cert que $f \in O(g)$ implica $\log f \in O(\log g)$? Justifiqueu la resposta o doneu un contraexemple.
- És cert que $f \in O(g)$ implica $2^f \in O(2^g)$? Justifiqueu la resposta o doneu un contraexemple.

Problema 5**(2,5 punts)**

Siguin $A = \langle a_1, a_2, \dots, a_n \rangle$ i $B = \langle b_1, b_2, \dots, b_m \rangle$ dues seqüències amb $n \geq m$. Es diu que B és una subseqüència de A , i s'escriu $B \subseteq A$, si existeixen m índexs en A , $i_1 < i_2 < \dots < i_m$, tals que $\forall j: 1 \leq j \leq m: b_j = a_{i_j}$. Per exemple, si $A = abcaddaa$ i $B = bcda$, llavors $B \subseteq A$, però si $B = cbda$, aleshores $B \not\subseteq A$.

Suposeu que disposeu d'un algorisme `is_subseq`, de cost $O(n + m)$, que determina si B és o no subseqüència de A .

Definim B^i , amb $i \geq 0$, com la seqüència que s'obté prenent un per un i de forma consecutiva els elements de B i repetint-los i vegades. Per exemple, si $B = abbc$, $B^0 = \lambda$ (seqüència buida), $B^1 = abbc$, $B^2 = aabbbcc$, $B^3 = aaabbbbbbccc$, etc.

És fàcil mostrar que si $B^i \subseteq A$, aleshores $\forall j: 0 \leq j \leq i: B^j \subseteq A$.

Proposeu un algorisme de dividir i vèncer per trobar, en temps $O(n \log n)$, el màxim valor de i tal que $B^i \subseteq A$. Justifiqueu tant la correctesa com el cost de la vostra solució.

Noteu que si B no és una subseqüència de A , llavors i serà zero, i que el valor més gran possible per a i és n/m , perquè per tal que B^i sigui subseqüència de A , la longitud de B^i ha de ser menor o igual a la de A .

Examen Parcial EDA

Duració: 2 hores

22/10/2012

Aquestes identitats us poden ser útils per aquest examen:

- $\sum_{i=1}^n i = n(n+1)/2$
- $\sum_{i=1}^n i^2 = n(n+1)(2n+1)/6$
- $\sum_{i=1}^n i^3 = (n(n+1)/2)^2$
- $\sum_{i=1}^n c^i = \Theta(c^n)$ per a $c > 1$.

Problema 1

(2,5 punts)

- Sabent que $T_1 = O(f)$ i que $T_2 = O(f)$, digueu si les afirmacions següents són certes o falses.

– (0,5 punts) $T_1 + T_2 = O(f)$.

– (0,5 punts) $T_1 - T_2 = O(f)$.

– (0,5 punts) $T_1 / T_2 = O(f)$.

– (0,5 punts) $T_1 = O(T_2)$.

- (0,5 punts) Resoleu la recurrència:

$$T(n) = \begin{cases} 1 & \text{si } n \leq 4 \\ 2T(\frac{n}{4}) + \sqrt{n} & \text{si } n > 4 \end{cases}$$

Problema 2

(1 punt)

Demostreu que el nombre de bits de $n!$ és $\Theta(n \log n)$.

Problema 3

(1,5 punts)

```
int misteri (int n) {
    int r = 0;
    for (int i = 1; i ≤ n; ++i)
        for (int j = i+1; j ≤ n; ++j)
            for (int k = j+1; k ≤ n; ++k)
                ++r;
    return r;
}
```

1. (1 punt) Assumint $n \geq 3$, aquesta funció calcula un polinomi $p(n) = a_0 + a_1n + a_2n^2 + \dots + a_d n^d$. Quin? Doneu-ne el grau d i tots els coeficients exactes a_0, \dots, a_d .
2. (0,5 punts) Doneu una fita ajustada (amb notació Θ) al cost de la funció *misteri*.

Problema 4**(2 punts)**

Considereu el codi següent:

```
double misteri_a(int n, double x) {
    if (n == 0) return 1;
    double aux = misteri_a(n/2, x);
    aux *= aux;
    if (n%2 == 0) return aux;
    return aux*x;
}

void misteri_b(int n, vector<double>& V) {
    for (int i = 0; i < int(V.size()); ++i) V[i] = misteri_a(n, V[i]);
}
```

Digueu, justificadament, què fan *misteri_a* i *misteri_b*, i quin cost tenen, en funció de n i $m = V.size()$, en els casos pitjor, mitjà i millor.

Problema 5**(1 punt)**

Proposeu (sense implementar) un algorisme eficient per calcular m^m , on m és un natural llarg. Recordeu que es poden fer servir algorismes explicats a classe. Doneu el cost del vostre algorisme en funció de la talla de l'entrada n , és a dir, $n = \log_2 m$.

Problema 6**(2 punts)**

Es diu que una funció $f: [a, b] \rightarrow \mathbb{R}$ és:

- *estrictament creixent* quan per a tot $x, y \in [a, b]$, $x < y$ implica $f(x) < f(y)$;
- *estrictament decreixent* quan per a tot $x, y \in [a, b]$, $x < y$ implica $f(x) > f(y)$;
- *estrictament monòtona* quan és estrictament creixent o decreixent.

Sigui $f: [a, b] \rightarrow \mathbb{R}$ una funció contínua i estrictament monòtona que satisfà que $f(a) \cdot f(b) < 0$. És sabut que aleshores existeix un únic $z \in (a, b)$, anomenat el *zero* de la funció, tal que $f(z) = 0$.

Assumint que f és una funció contínua i estrictament monòtona $f: [a, b] \rightarrow \mathbb{R}$ tal que $f(a) \cdot f(b) < 0$, escriviu en C++ una funció

```
double zero (double a, double b, double tol)
```

que, donades a, b i una tolerància tol tal que $0 < tol < b - a$, retorni un \tilde{z} tal que el zero de f es troba en l'interval $(\tilde{z} - \frac{tol}{2}, \tilde{z} + \frac{tol}{2})$. Analitzeu el cost del vostre programa en el cas pitjor. Assumiu que podeu fer crides a f , i que el seu cost és $O(1)$. Només es consideraran com a vàlides les solucions amb cost $O(\log(\frac{b-a}{tol}))$.

Examen Parcial EDA**Duració: 2 hores****18/3/2013****Problema 1****(2 punts)**

Disposem dels tres algorismes següents per resoldre un cert problema (n denota la mida de l'entrada):

- A: resol el problema dividint-lo en 5 subproblemes de mida $n/2$, resol recursivament cada subproblema i combina les solucions en temps $\Theta(n)$.
- B: resol el problema dividint-lo en 2 subproblemes de mida $n - 1$, resol recursivament cada subproblema i combina les solucions en temps $\Theta(1)$.
- C: resol el problema dividint-lo en 9 subproblemes de mida $n/3$, resol recursivament cada subproblema i combina les solucions en temps $\Theta(n^2)$.

Quin és el cost en temps de cada algorisme fent servir la notació Θ ?

(0,5 punts) Algorisme A:

(0,5 punts) Algorisme B:

(0,5 punts) Algorisme C:

(0,5 punts) Quin dels tres algorismes escolliries? Per què?

Problema 2**(2 punts)**

Sigui $\pi(n)$ el nombre de nombres primers dins l'interval $[1, \dots, n]$. És un fet que

$$\pi(n) = \Theta\left(\frac{n}{\log n}\right).$$

Per a les preguntes següents, recordeu que els nombres *compostos* són els que no són primers, i que els nombres *quadrats* són els de la forma k^2 on k és un natural.

(1 punt) Digueu si és CERT o FALS que, per a n suficientment gran, hi ha més nombres compostos que primers dins l'interval $[1, \dots, n]$, i justifiqueu la resposta:

(1 punt) Digueu si és CERT o FALS que, per a n suficientment gran, hi ha més nombres quadrats que primers dins l'interval $[1, \dots, n]$, i justifiqueu la resposta:

Problema 3**(2 punts)**

Donat un vector V amb n nombres diferents $V[0], \dots, V[n-1]$, i un nombre s , es vol saber si existeixen tres índexs i, j i k entre 0 i $n-1$ tals que $V[i] + V[j] + V[k] = s$. Els índexs poden estar repetits. Per exemple, si $n = 2$, $V[0] = 5$, $V[1] = 6$, i $s = 16$, la resposta és afirmativa. (Una solució seria $i = j = 0$, $k = 1$.)

Considereu aquests algorismes:

- a) Provem per a tota i , tota j i tota k entre 0 i $n-1$, si $V[i] + V[j] + V[k] = s$.
- b) Provem per a tota i entre 0 i $n-1$, tota j entre i i $n-1$ i tota k entre j i $n-1$, si $V[i] + V[j] + V[k] = s$.

- c) Construïm un vector Q de mida n^2 amb totes les sumes $V[i] + V[j]$, amb i i j entre 0 i $n - 1$. Ordenem Q . Busquem, per a tota k entre 0 i $n - 1$, si $s - V[k]$ es troba a Q .
- d) Construïm un vector Q de mida n^2 amb totes les sumes $V[i] + V[j]$, amb i i j entre 0 i $n - 1$. Ordenem V . Busquem, per a tota ℓ entre 0 i $n^2 - 1$, si $s - Q[\ell]$ es troba a V .

Digueu raonadament el cost en el cas pitjor de cadascun d'aquests algorismes. (0,5 punts cadascun).

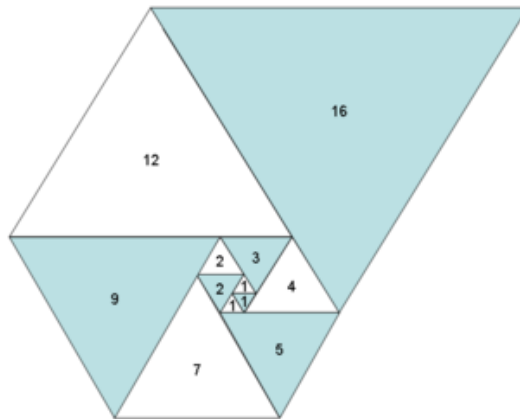
Problema 4

(2 punts)

La *successió de Padovan* és una successió de nombres naturals definida pels valors inicials $P(0) = P(1) = P(2) = 1$ i la relació de recurrència

$$P(n) = P(n - 2) + P(n - 3)$$

per a $n \geq 3$. Entre altres llocs, aquesta successió apareix, gràficament, aquí:



1. Implementa en C++ una funció `int g1(int n)` que calculi el n -èsim nombre de Padovan. La funció ha de tenir cost en temps $O(n)$.
2. Es pot comprovar que

$$\begin{pmatrix} P(n) \\ P(n-1) \\ P(n-2) \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

per $n \geq 2$. Fent ús d'aquest fet (que no us demanem demostrar), i d'un tipus `matrix<int>` amb constructor `matrix<int>(files, columnes, valor)` i amb operació de multiplicació $m1 * m2$, completa la següent implementació de `int g2(int n)` que també calcula $P(n)$, aquest cop en temps $O(\log n)$.

```
void misteri (const matrix<int>& m, int k, matrix<int>& p) {
    if (  )
        for (int i = 0; i < 3; ++i) p[i][i] = 1;
    else {
        misteri (  );
        p =  *  ;
    }
}
```

```

        if (k % 2 == 1)  ;
    } }

int g2(int n) {
    if (n ≤ 2) return 1;
    else {
        matrix<int> m(3, 3, 0);
        m[0][1] = m[0][2] = m[1][0] = m[2][1] = 1;
        matrix<int> p(3, 3, 0);
        mister (m, n-2, p);
        return p[0][0] + p[0][1] + p[0][2];
    } }

```

L'enunciat d'aquest problema està inspirat en la pàgina: http://en.wikipedia.org/wiki/Padovan_sequence

Problema 5

(2 punts)

Suposeu donat un tipus `vector_inf<int>` que implementa vectors de mida ∞ en què les n primeres posicions (de la 0 a la $n - 1$) contenen valors enters (i per tant finits) no necessàriament ordenats, i que les posicions restants (a partir de la n) contenen el valor ∞ . Doneu, en C++, un algorisme

```
int busca( vector_inf<int>& V)
```

que, donat un vector V com aquest amb $n \geq 1$, determini n en temps $O(\log n)$.

Suposeu que el tipus `vector_inf<int>` està implementat de tal manera que podeu accedir a les seves posicions de la mateixa manera que amb els vectors ordinaris en temps constant. També podeu fer comparacions $v[i] == \infty$ en temps constant.

Pista: comenceu per trobar un ∞ ràpidament.

Examen Parcial EDA

Duració: 2 hores

21/10/2013

Problema 1

(2 punts)

- Sigui n parell. El cost de l'algorisme d'ordenació per inserció amb l'entrada

$$[2, 1, 4, 3, 6, 5, \dots, 2i, 2i - 1, \dots, n, n - 1]$$

és $T(n) = \Theta(\quad)$.

- Indiqueu **totes** les afirmacions que siguin correctes: El cost de mergesort en el cas pitjor és: ☐ $O(n^2)$; ☐ $\Theta(n \log n)$; ☐ $\Omega(n \log n)$.
- Doneu una funció $f(n)$ que sigui alhora $\Omega(n)$ i $O(n \log n)$, però que no sigui Θ de cap de les dues. Resposta: $f(n) = \quad$
- Resoleu $T(n) = 3T(n/3) + \Theta(n^2)$. Resposta: $T(n) = \Theta(\quad)$.
- La **recurrència** que expressa el cost de l'algorisme de Karatsuba per multiplicar dos nombres de n dígitos és $T(n) = \quad$.
- Sigui x un vector de n bits qualssevol. Interpretant x com un nombre natural escrit en binari, el cost en cas pitjor de l'algorisme obvi per sumar 1 a x és $T(n) = \Theta(\quad)$.
- Sigui x un vector de n bits escollits uniformement i independentment a l'atzar. Interpretant x com un nombre natural escrit en binari, el cost esperat de l'algorisme obvi per sumar 1 a x és $T(n) = \Theta(\quad)$.
- Considereu l'algorisme següent de cost $\Theta(\log n)$ on n és la mida del vector:

```

int dicotomica(vector<int>& v, int e, int d, int x) {
    if (e > d) return -1;
    int m = (d + e)/2;
    if (v[m] < x) return dicotomica(v, m + 1, d, x);
    if (v[m] > x) return dicotomica(v, e, m - 1, x);
    return m;
}

```

Quin cost tindria aquest mateix algorisme si tinguéssim la mala pata d'oblidar-nos l'& en el pas de paràmetres? Resposta: $T(n) = \Theta(\quad)$.

Problema 2

(2 punts)

Donat un vector A amb $n > 0$ nombres, considerem dos algorismes per calcular els elements mínim (\min) i màxim (\max) simultàniament:

1. **Seqüencial.** Les variables \min i \max prenen inicialment el valor $A[0]$ i es van actualitzant en un recorregut seqüencial del vector.
2. **Dividir i vèncer.** Si el vector només té un o dos elements, es calculen \min i \max de la manera òbvia fent una sola comparació. En cas que $n > 2$, es divideix el vector en dues meitats i, per a cadascuna, es fa una crida recursiva i s'assigna a \min el més petit dels mínims fent una comparació i a \max el més gran dels màxims fent una altra comparació.

Justifiqueu les afirmacions següents:

(1 punt) El cost asimptòtic de tots dos algorismes és el mateix.

(1 punt) Si comptem el nombre **exacte** de comparacions entre elements del vector que fa cada algorisme, el del segon és millor que el del primer. Per simplificar, suposeu que n és una potència de 2. (Pista: per analitzar el segon algorisme, dibuixeu l'arbre de recursió per $n = 8$ i recordeu que $\sum_{i=0}^{k-1} 2^i = 2^k - 1$.)

Problema 3

(2 punts)

(1 punt) Proposeu un algorisme lineal que, donat un vector $V[1, \dots, n]$ que conté una permutació qualsevol dels nombres de 1 a n , retorni un vector $D[1, \dots, n-1]$ on $D[i]$ és la distància a la que es troben els nombres i i $i+1$ en el vector V . Per exemple, si $V = [5, 2, 1, 4, 6, 3]$, llavors $D = [1, 4, 2, 3, 4]$. No es valorarà cap solució que no sigui de cost $\Theta(n)$.

(1 punt) Ara suposeu que el vector $V[1, \dots, n]$ conté nombres diferents qualssevol (no necessàriament entre 1 i n) i voleu retornar un vector $D[1, \dots, n-1]$ on $D[i]$ és la distància a la que es troben l' i -èssim nombre de V i l' $(i+1)$ -èssim nombre de V , enumerats de petit a gran. Per exemple, si $V = [1231, -2, 453456, -23434]$, llavors $D = [2, 1, 2]$. Descriviu un algorisme de cost $\Theta(n \log n)$ per fer això. (Pista: penseu en una modificació del mergesort.)

Problema 4

(2 punts)

Considereu el codi següent:

```
double misteri_a(int n, double x) {
    if (n == 0) return 1;
    double aux = misteri_a(n/2, x);
    aux *= aux;
    if (n%2 == 0) return aux;
    return aux*x;
}

void misteri_b(int n, vector<double>& V) {
    for (int i = 0; i < int(V.size()); ++i) V[i] = misteri_a(n, V[i]);
}
```

Digueu, justificadament, què fan *misteri_a* i *misteri_b*, i quin cost tenen, en funció de n i $m = V.size()$, en els casos pitjor, mitjà i millor.

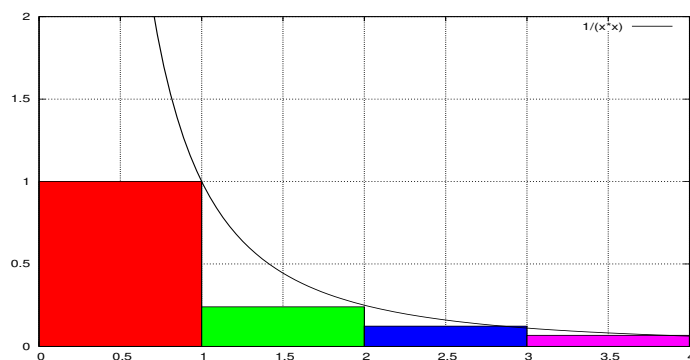
Problema 5

(1 punt)

Sigui

$$S(n) = \sum_{i=1}^n \frac{1}{i^2}.$$

Inspirats en el gràfic adjunt, demostreu que $S(n) = \Theta(1)$. (Pista: àrea = integral.)

**Problema 6****(1 punt)**

Demostreu o doneu un contraexemple a l'enunciat següent: Per qualsevol funció $f(n)$ i qualsevol constant $c > 0$ tenim que $f(cn) = \Theta(f(n))$.

Examen Parcial EDA

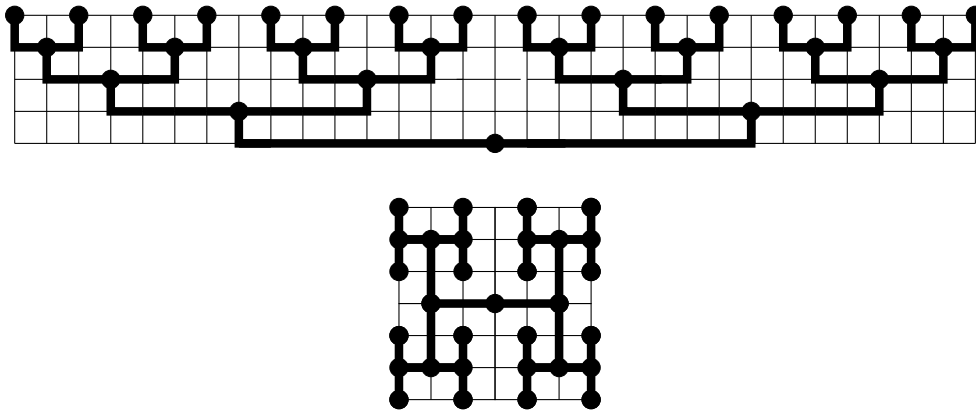
Duració: 2 hores

24/3/2014

Problema 1

(3 punts)

Com a grafs, tots els arbres són planars. En particular això vol dir que tots es poden representar en una graella rectangular prou gran de manera que les arestes no s'intersequin les unes amb les altres (excepte, òbviament, en els vèrtexs). Aquí teniu dues representacions planars d'un arbre binari complet de 16 fulles:



(1 punt) Seguint el patró recursiu de la **primera** representació, doneu les recurrències que expressen l'alçada $H(n)$ i l'amplada $W(n)$, respectivament, de la graella que es necessita per representar un arbre binari complet de 2^n fulles en aquesta representació.

(1 punt) Seguint el patró recursiu de la **segona** representació, doneu la recurrència que expressa la longitud $L(n)$ de la graella quadrada que es necessita per representar un arbre binari complet de 4^n fulles en aquesta construcció.

(1 punt) Si el vostre objectiu és minimitzar l'àrea de la graella per un arbre binari complet de 4^n fulles, expliqueu quina de les dues construccions escollireu (per relacionar les dues construccions noteu que $4^n = 2^{2n}$). No cal que resolcu les recurrències de forma exacta; una estimació asimptòtica és suficient.

Problema 2

(2 punts)

Considereu els quatre codis a continuació:

```
void f1 (int n) {
    int x = 2;
    int y = 1;
    while (y ≤ n) {
        y = y + x;
        x = x + 1;
    }
}
```

```
void f2 (int n) {
    int x = 2;
    int y = 1;
    while (y ≤ n) {
        y = y + x;
        x = 2*x;
    }
}
```

```
void f3 (int n) {
    int x = 2;
    int y = 1;
```

```
void f4 (int n) {
    int x = 2;
    int y = 1;
```

<pre>while (y ≤ n) { y = y*x; x = 2*x; }</pre>	<pre>while (y ≤ n) { y = y*x; x = x*x; }</pre>
--	--

Quin cost té f_1 ? Resposta: $\Theta(\quad)$

Quin cost té f_2 ? Resposta: $\Theta(\quad)$

Quin cost té f_3 ? Resposta: $\Theta(\quad)$

Quin cost té f_4 ? Resposta: $\Theta(\quad)$

Una resposta correcta sense una justificació adequada **no rebrà cap punt**.

Problema 3

(3 punts)

Una *inversió* en un vector d'enters $T[0 \dots n-1]$ és un parell de posicions del vector en desordre, és a dir, un parell (i, j) amb $0 \leq i < j < n$ tal que $T[i] > T[j]$.

(1,5 punts) Sigui $T[0 \dots n-1]$ un vector d'enters. Demostreu que si en T hi ha una inversió (i, j) , aleshores T té almenys $j - i$ inversions.

(1,5 punts) Fixem un valor N tal que $N \geq 0$ (és a dir, N en aquest exercici és una constant). Usant Divideix i Venceràs, implementeu en C++ una funció

bool cerca(int x, const vector<int>& T)

que, donats un enter x i un vector d'enters T de mida n amb com a molt N inversions, digui si x és a T . La funció ha de tenir cost en el cas pitjor $\Theta(\log n)$. Demostreu que té el cost desitjat.

Pista: per respondre l'apartat 2, useu l'apartat 1. A més, considereu com a cas base el tractament de subvectors de mida $\leq 2N$, i com a cas recursiu el complementari.

Problema 4

(2 punts)

Quatre preguntes curtes:

1. Sigui $f(n) = n \log(\cos(n\pi) + 4)$. Llavors $f(n) = \Theta(\quad)$.
2. Quin és l'últim dígit de 7^{1024} escrit en decimal? Resposta: \quad .
3. Quina és la **recurrència** que expressa el cost de l'algorisme de Strassen per multiplicar matrius $n \times n$. Resposta $T(n) = \quad$.
4. Un algorisme pot rebre els 2^n vectors de n bits com entrada. En $2^n - 1$ d'aquestes entrades el cost de l'algorisme és $\Theta(n^2)$ i en l'entrada restant el seu cost és $\Theta(n^4)$. Per

tant, el cost en el cas pitjor és $\Theta(n^4)$ i el cost en el cas millor és $\Theta(n^2)$. Quin és el cost en el cas mitjà quan l'entrada s'escull uniformement a l'atzar (i per tant cada entrada té probabilitat $1/2^n$)?

Resposta: $T(n) = \Theta(\text{ })$.

Justificacions per les preguntes 1, 2 i 4 (per la pregunta 3 **no** cal justificació):

Examen Parcial EDA

Duració: 2.5 hores

13/10/2014

Problema 1

(2 punts)

(1 punt) Demostreu que $\sum_{i=0}^n i^3 = \Theta(n^4)$.

Ajut: Hi ha diverses maneres de fer-ho, una d'elles demostrant l' O i l' Ω per separat.

(1 punt) Sigui $f(n) = 2\sqrt{\log n}$ i $g(n) = n$. Asimptòticament, quina creix més ràpid? Demostreu-ho.

Problema 2

(3 punts)

Considerem el procediment *misteri* que rep com entrada un vector A i un enter positiu k de manera que tots els elements de A estan entre 0 i k , ambdós inclosos.

```
void misteri (const vector<int>& A, vector<int>& B, int k){
    int n = A.size ();
    vector<int> C(k+1, 0);
    for (int j = 0; j < n; ++j) ++C[A[j]];
    for (int i = 1; i ≤ k; ++i) C[i] += C[i-1];
    B = vector<int>(n);
    for (int j = n-1; j ≥ 0; --j){
        B[C[A[j]] - 1] = A[j];
        --C[A[j]];
    }
}
```

(a) (1 punt) Què conté B al finalitzar l'execució de *misteri*?

(b) (1 punt) Si n és la longitud del vector A i us asseguruen que $k = O(n)$, quin és el cost asimptòtic de *misteri* en funció de n ?

(c) (1 punt) Sense escriure codi, descriu un algorisme per al problema següent: donat un enter positiu k , un vector A de n enters entre 0 i k , ambdós inclosos, i un vector P de m parells d'enters $(a_0, b_0), \dots, (a_{m-1}, b_{m-1})$, amb $0 \leq a_i \leq b_i \leq k$ per cada $i = 0, \dots, m-1$, cal escriure el nombre d'elements de A que es troben en l'interval $[a_i, b_i]$ per $i = 0, \dots, m-1$ en temps $\Theta(m + n + k)$, i en particular temps $\Theta(m + n)$ quan $k = O(n)$.

Problema 3

(3 punts)

Un joc de taula d'atzar té 64 posicions possibles $1, \dots, 64$. Les regles del joc són tals que, des de cada posició $i \in \{1, \dots, 64\}$, en una tirada podem anar a parar a qualsevol altra posició $j \in \{1, \dots, 64\}$ amb probabilitat $P_{i,j}$. Sigui $P = (P_{i,j})_{1 \leq i,j \leq 64}$ la matriu de probabilitats.

(a) (1 punt) Recordeu que si $R = P^2$, llavors $R_{i,j} = \sum_{k=1}^{64} P_{i,k} P_{k,j}$ per cada i, j . Sabent que, com diu l'enunciat, $P_{i,j}$ és la probabilitat d'anar a parar a j des de i en una tirada, com interpreteu $R_{i,j}$?

(b) (2 punts) Dissenyeu un algorisme que, donada la matriu de probabilitats $P = (P_{i,j})_{1 \leq i,j \leq 64}$ i donat un nombre de moviments $t \geq 0$, calculi la matriu $(Q_{i,j})_{1 \leq i,j \leq 64}$ on cada $Q_{i,j}$ és la probabilitat que el joc acabi a la posició j després de jugar exactament t tirades, començant des de la posició i .

```
typedef vector<double> Fila;
typedef vector<Fila> Matriu;
```

```
void probabilitats (const Matriu & P, int t, Matriu & Q)
```

Nota 1: Per obtenir la nota màxima, el vostre algorisme ha de tenir cost $\Theta(\log t)$.

Nota 2: Si us cal alguna funció auxiliar, implementeu-la al darrera.

Nota 3: Justifiqueu el cost al darrera.

Problema 4

(2 punts)

(a) (1 punt) Suposem que un algorisme pot rebre qualsevol dels 2^n vectors de n bits amb igual probabilitat. Suposem que el cost de l'algorisme en el cas pitjor és $\Theta(n^2)$ i que el cost en el cas millor és $\Theta(n)$. Quantes entrades cal que provoquin cost $\Omega(n^2)$ per estar segurs que l'algorisme tindrà cost $\Theta(n^2)$ en el cas mitjà?

(b) (0.5 punts) Considereu el següent algorisme per sumar una unitat a un nombre natural representat per un vector `vector<int> A` de n dígit decimal:

```
int i = n - 1;
while (i ≥ 0 and A[i] == 9) { A[i] = 0; --i; }
if (i ≥ 0) ++A[i]; else cout << "Overflow" << endl;
```

Si cada dígit $A[i]$ de l'entrada és equiprobable i independent de la resta (és a dir, cada $A[i]$ és un dels 10 dígit possibles amb probabilitat $1/10$ sense tenir en compte la resta), quina és la probabilitat que aquest algorisme faci exactament k iteracions quan $0 \leq k \leq n - 1$?

(c) (0.5 punts) Apliqueu el teorema mestre de recurrències substractores per resoldre la recurrència $T(n) = \frac{1}{10}T(n-1) + \Theta(1)$ amb el cas base $T(0) = \Theta(1)$.

————— FI DE L'EXAMEN —————

(d) (extra bonus **opcional**: 1 punt addicional a la nota) A què correspon la recurrència $T(n)$ de l'apartat (c) en relació a l'algorisme de l'apartat (b)?

Examen Parcial EDA

Duració: 2h30m

23/3/2015

Problema 1: Anàlisi de costos

(2 punts)

El garbell d'Eratòstenes (276–194 aC) és un mètode per generar tots els nombres primers més petits o iguals que un n donat. El mètode fa així: recorrent la seqüència dels nombres $2, 3, 4, \dots, n$, busca el següent nombre x que no estigui marcat, marca tots els seus múltiples $2x, 3x, 4x, \dots$ més petits o iguals que n , i torna a començar. Quan acaba, els $x \geq 2$ que no estan marcats són els nombres primers. En C++:

```
vector<bool> M(n + 1, false);
for (int x = 2; x ≤ n; ++x) {
    if (not M[x]) {
        for (int y = 2*x; y ≤ n; y += x) M[y] = true;
    }
}
```

Per a les tres primeres preguntes es demana una expressió exacta (no asimptòtica) *en funció de n* . Si us cal feu servir la notació $\lfloor z \rfloor$ que arrodoneix z al màxim enter més petit o igual que z ; per exemple $\lfloor \pi \rfloor = \lfloor 3.14\dots \rfloor = 3$.

(a) (0.33 punts) Quantes vegades s'executarà $M[y] = \text{true}$ quan x valgui 2?

Resposta: Exactament vegades.

(b) (0.34 punts) Quantes vegades s'executarà $M[y] = \text{true}$ quan x valgui 15?

Resposta: Exactament vegades.

(c) (0.33 punts) Quantes vegades s'executarà $M[y] = \text{true}$ quan x valgui 17?

Resposta: Exactament vegades.

(d) (0.5 punts) Se sap que

$$\sum_{\substack{p=2 \\ p \text{ primer}}}^n \frac{1}{p} = \Theta(\log \log n).$$

Feu servir això i les respostes de les preguntes anteriors per calcular el cost de l'algorisme en funció de n , en notació asimptòtica. Resposta: $\Theta(\text{ })$. Justificació:

(e) (0.5 punts) Una millora seria substituir la condició $x \leq n$ del bucle extern per $x*x \leq n$. Millora això el cost asimptòtic? Resposta i justificació:

Problema 2: Strassen & company

(2 punts)

L'algorisme escolar per multiplicar matrius $n \times n$ fa $\Theta(n^3)$ operacions aritmètiques. L'any 1969 Strassen va trobar un algorisme que fa $\Theta(n^{2.81})$ operacions aritmètiques. Vint-i-un anys més tard, Coppersmith i Winograd van descobrir un mètode que fa $\Theta(n^{2.38})$ operacions.

Suposant (per simplificar) que les constants implícites en la notació Θ són 1, 10 i 100, respectivament, i que s'apliquen a cada $n \geq 1$ (és a dir, que els costos són n^3 , $10n^{2.81}$ i $100n^{2.38}$,

respectivament, per a tot $n \geq 1$), calculeu el mínim n a partir del qual un d'aquests algorismes fa menys operacions que un altre.

(a) (1 punt) Per a $n \geq \boxed{}$, Strassen millora l'escolar.

(b) (1 punt) Per a $n \geq \boxed{}$, Coppersmith-Winograd millora Strassen.

Justificacions:

Nota: Si no porteu calculadora (estàveu avisats!), deixeu indicada la solució en forma d'expressió aritmètica.

Problema 3: Un de dissenyar algorismes

(3 punts)

Donada una seqüència de n intervals no buits $[a_1, b_1], \dots, [a_n, b_n]$, volem calcular, en temps $O(n \log n)$, la seva unió representada com una seqüència d'intervals disjunts ordenada segons l'extrem esquerre dels intervals. Per exemple, si els intervals de l'entrada són

$$[17, 19] \quad [-3, 7] \quad [4, 9] \quad [18, 21] \quad [-4, 15]$$

llavors la sortida ha de ser $[-4, 15] \quad [17, 21]$.

(a) (1 punt) Descriviu un algorisme que resolgui aquest problema. Expliqueu l'algorisme en paraules, sense escriure codi, però de manera prou clara perquè es pugui implementar. Supposeu que l'entrada ve donada pels vectors (a_1, \dots, a_n) i (b_1, \dots, b_n) , amb $a_i \leq b_i$ per a tot $i = 1, \dots, n$.

(b) (1 punt) En aquest apartat, a més de la seqüència de n intervals, ens donen una seqüència de m reals diferents p_1, \dots, p_m i volem determinar quants cauen en algun interval de la unió (només ens cal el número; no pas quins són). Fent servir l'algorisme de l'apartat anterior, descriviu un algorisme que resolgui aquest problema en temps $O(n \log n)$ quan $m = n$. Supposeu que l'entrada ve donada pels vectors a i b de l'apartat anterior, i pel vector (p_1, \dots, p_m) amb $p_i \neq p_j$ si $i \neq j$.

(c) (1 punt) Si sabéssiu que m està fitat per una constant petita i independent de n , per exemple $m \leq 5$, faríeu servir el mateix algorisme? Si no, quin faríeu servir? Si decidiu canviar d'algorisme, expliciteu-ne el cost.

Problema 4: Preguntes curtes

(3 punts)

- (0,5 punts) Determineu si són iguals ($=$) o diferents (\neq) i demostreu-ho:

$$\Theta(3^{\log_2(n)}) \quad \boxed{} \quad \Theta(3^{\log_4(n)}).$$

- (0,5 punts) Calculeu $2^1 \cdot 2^2 \cdot 2^3 \cdot \dots \cdot 2^{99} \cdot 2^{100} \pmod{9}$. Aquest problema no està pensat per fer amb calculadores.
- (1 punt) Ordeneu les funcions següents segons el seu ordre de creixement asimptòtic: $n^4 - 3n^3 + 1$, $(\ln(n))^2$, \sqrt{n} , $n^{1/3}$. Excepcionalment, no cal que ho justifiqueu.

- (1 punt) Considereu les tres alternatives següents per resoldre un mateix problema:
 - A: divideix una instància de talla n en cinc instàncies de talla $n/2$, resol recursivament cada instància, i combina les solucions en temps $\Theta(n)$.
 - B: donada una instància de talla n , resol recursivament dues instàncies de talla $n - 1$ i combina les solucions en temps constant.
 - C: divideix una instància de talla n en nou instàncies de talla $n/3$, resol recursivament cada instància, i combina les solucions en temps $\Theta(n^2)$.

Escriuiu les recurrències corresponents i resoleu-les. Quina alternativa és la més eficient?

Examen Parcial EDA

Duració: 2.5 hores

19/10/2015

Problema 1

(3.5 punts)

Els nombres de Fibonacci estan definits per la recurrència $f_k = f_{k-1} + f_{k-2}$ per a $k \geq 2$, amb $f_0 = 0$ i $f_1 = 1$. Responen els següents apartats:

- (a) (0.5 punts) Considereu la següent funció *fib1* que donat un enter no negatiu k retorna f_k :

```
int fib1 (int k) {  
    vector<int> f(k+1);  
    f[0] = 0;  
    f[1] = 1;  
    for (int i = 2; i ≤ k; ++i)  
        f[i] = f[i-1] + f[i-2];  
    return f[k];  
}
```

Descriviu de la manera més precisa possible el cost asimptòtic *en temps i en espai* de *fib1* (k) en funció de k .

- (b) (1 punt) Demostreu que, per a $k \geq 2$, se satisfà la identitat matricial següent:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k-1} = \begin{pmatrix} f_k & f_{k-1} \\ f_{k-1} & f_{k-2} \end{pmatrix}$$

- (c) (1 punt) Completeu els blancs del codi a continuació per tal que la funció *fib2* (*k*) calculi f_k , donat un $k \geq 0$.

```
typedef vector<vector<int>> matrix;

matrix mult(const matrix& A, const matrix& B) {
    // Pre: A i B són matrius quadrades de les mateixes dimensions
    int n = A.size ();
    matrix C(n, vector<int>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                
    return C;
}

matrix misteri (const matrix& M, int q) {
    int s = M.size ();
    if (q == 0) {
        matrix R(s, vector<int>(s, 0));
        for (int i = 0; i < s; ++i) 
        return R;
    }
    else {
        matrix P = misteri (M, q/2);
```

```

    if (  ) return mult(P, P);
    else return  ;
} }

```

```

int fib2 (int k) {
    if (k ≤ 1) return k;
    matrix M = { {1, 1}, {1, 0} };
    matrix P = misteri (M, k-1);
    return  ;
}

```

- (d) (1 punt) Descriu de la manera més precisa possible el cost asimptòtic en temps de $fib2(k)$ en funció de k .

Problema 2

(3.25 punts)

Donats un vector d'enters v i un enter x , la funció

```

int posicio (const vector<int>& v, int x) {
    int n = v.size ();
    for (int i = 0; i < n; ++i)
        if (v[i] == x)
            return i;
    return -1;
}

```

examina les $n = v.size()$ posicions de v i retorna la primera que conté x , o -1 si no n'hi ha cap.

- (a) (0.75 punts) Descriviu en funció de n el cost asimptòtic en temps de *posicio* en el cas millor de la forma més precisa possible. Quan es pot donar aquest cas millor?

- (b) (0.75 punts) Descriviu en funció de n el cost asimptòtic en temps de *posicio* en el cas pitjor de la forma més precisa possible. Quan es pot donar aquest cas pitjor?

- (c) (0.75 punts) Demostreu que per a tot enter $n \geq 1$, es compleix:

$$\sum_{i=1}^n \frac{i}{2^i} = 2 - \frac{n}{2^n} - \frac{1}{2^{n-1}}.$$



- (d) (1 punt) Suposem que tenim una distribució de probabilitat sobre els paràmetres d'entrada. Concretament, la probabilitat que x sigui l'element $v[i]$ és $\frac{1}{2^{i+1}}$ per a $0 \leq i < n - 1$, i que sigui l'element $v[n - 1]$ és $\frac{1}{2^{n-1}}$. En particular, aquestes probabilitats sumen 1, de manera que x sempre és un dels n valors de v .

Descriviu en funció de n el cost asimptòtic en temps de *posicio* en el cas mig de la forma més precisa possible.

Problema 3**(3.25 punts)**

En aquest exercici abordarem el problema de, donats dos enters positius a i b , calcular el seu màxim comú divisor $\gcd(a, b)$. Recordeu que $\gcd(a, b)$ és, per definició, l'únic enter positiu g tal que:

1. $g \mid a$ (g divideix a),
2. $g \mid b$,
3. si $d \mid a$ i $d \mid b$, llavors $d \mid g$.

(a) (1.25 punts) Demostreu que les identitats següents són certes:

$$\gcd(a, b) = \begin{cases} 2\gcd(a/2, b/2) & \text{si } a, b \text{ són parells} \\ \gcd(a, b/2) & \text{si } a \text{ és senar i } b \text{ és parell} \\ \gcd((a-b)/2, b) & \text{si } a, b \text{ són senars i } a > b \end{cases}$$

Pista: podeu fer servir que donats dos enters positius a i b tals que $a > b$, es té $\gcd(a, b) = \gcd(a - b, b)$.

- (b) (1 punt) Escriuiu una funció `int gcd(int a, int b)` en C++ que, usant `divideix` i `vençeràs` i l'apartat (a), calculi el màxim comú divisor $\gcd(a, b)$ de dos enters positius a, b donats.

Pista: podeu fer servir també que per tot enter positiu a , $\gcd(a, a) = a$.

- (c) (1 punt) Suposant que a i b són enters positius representats cadascun amb un vector de n bits, descriu de la manera més precisa possible el cost en temps en el cas pitjor de $\gcd(a, b)$ en funció de n . Quan es pot donar aquest cas pitjor?

Assumiu que el cost de les següents operacions amb enters de n bits: sumar, restar, comparar, multiplicar/dividir per 2 és $\Theta(n)$, i que calcular el residu mòdul 2 triga temps $\Theta(1)$.

Examen Parcial EDA

Duració: 2.5 hores

31/03/2016

Problema 1

(1 punt)

Responen les següents qüestions. En aquest exercici, **no** cal justificar les respostes.

1. (0.2 punts) El cost de l'algorisme d'Strassen per multiplicar dues matrius de mida $n \times n$ és $\Theta(\text{ })$

2. (0.6 punts) El teorema mestre de recurrències substractores diu que donada una recurrència $T(n) = aT(n - c) + \Theta(n^k)$ amb $a, c > 0$ i $k \geq 0$ llavors:

$$T(n) = \begin{cases} \text{ } & \text{si } a < 1, \\ \text{ } & \text{si } a = 1, \\ \text{ } & \text{si } a > 1. \end{cases}$$

3. (0.2 punts) L'algorisme d'ordenació mergesort usa $\Theta(\text{ })$ espai auxiliar per ordenar un vector de mida n .

Problema 2

(3 punts)

En aquest problema només es consideren els costos *en temps*. Considereu el següent programa:

```
void f(int m);
void g(int m);

void h(int n) {
    int p = 1;
    for (int i = 1; i ≤ n; i++) {
        f(i);
        if (i == p) {
            g(n);
            p *= 2;
        }
    }
}
```

- (a) (1.5 punts) Responen: si tant el cost de $f(m)$ com el de $g(m)$ és $\Theta(m)$, llavors el cost de $h(n)$ en funció de n és $\Theta(\text{ })$

Justificació:

(b) (1.5 punts) Responen: si el cost de $f(m)$ és $\Theta(1)$ i el cost de $g(m)$ és $\Theta(m^2)$, llavors el cost de $h(n)$ en funció de n és $\Theta(\text{ })$

Justificació:

Problema 3**(3 punts)**

Es diu que una matriu quadrada M de nombres enters de mida $n \times n$ és *simètrica* si $M_{i,j} = M_{j,i}$ per qualssevol i, j tals que $0 \leq i, j < n$.

Considerem la següent implementació de matrius simètriques amb vectors unidimensionals, que només guarda el “triangle inferior” per minimitzar l’espai consumit:

$$\begin{pmatrix} M_{0,0} & M_{0,1} & M_{0,2} & \dots & M_{0,n-1} \\ M_{1,0} & M_{1,1} & M_{1,2} & \dots & M_{1,n-1} \\ M_{2,0} & M_{2,1} & M_{2,2} & \dots & M_{2,n-1} \\ \vdots & & \ddots & & \\ M_{n-1,0} & M_{n-1,1} & M_{n-1,2} & \dots & M_{n-1,n-1} \end{pmatrix}$$

$$\Downarrow$$

$M_{0,0}$	$M_{1,0}$	$M_{1,1}$	$M_{2,0}$	$M_{2,1}$	$M_{2,2}$	\dots	$M_{n-1,0}$	$M_{n-1,1}$	$M_{n-1,2}$	\dots	$M_{n-1,n-1}$
-----------	-----------	-----------	-----------	-----------	-----------	---------	-------------	-------------	-------------	---------	---------------

Per exemple, la matriu simètrica 3×3

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

s'implementaria amb el vector unidimensional

1	2	-2	0	3	-1
---	---	----	---	---	----

- (a) (0.5 punts) Responen: el cost en espai d'aquesta representació per a una matriu simètrica $n \times n$ en funció de n és $\Theta(\text{ })$.

Justificació:

- (b) (1 punt) Donats $n > 0$ i un valor k , la funció

`pair<int,int> fila_columna(int n, int k);`

retorna el parell (i, j) on i és la fila i j és la columna del coeficient apuntat per k en un vector unidimensional que implementa una matriu simètrica $n \times n$. Si k no és un índex vàlid, retorna $(-1, -1)$. Per exemple, `fila_columna(3,2)` retorna $(1,1)$, `fila_columna(3,3)` retorna $(2,0)$, i `fila_columna(3,6)` retorna $(-1, -1)$.

Completeu la següent implementació de `fila_columna`:

```
int p(int x) { return  ;}

int misteri(int k, int l, int r) {
    if (l+1 == r) return l;
    int m = (l+r)/2;
    if (p(m) ≤ k) return misteri(k, , r);
    else return misteri(k, l, );
}

pair<int,int> fila_columna(int n, int k) {
    if (k < 0 or k ≥ p(n)) return  ;
    int i = misteri(k, 0, n);
    return {i,  };
}
```

- (c) (1 punt) Analitzeu el cost en temps en el cas pitjor de la implementació de `fila_columna(int n, int k)` de l'apartat anterior en funció de n .

- (d) (0.5 punts) Usant les funcions **double** *sqrt*(**double** *x*) i **int** *floor* (**double** *x*) que respectivament calculen \sqrt{x} i $\lfloor x \rfloor$ en temps $\Theta(1)$, doneu una implementació alternativa de *fila_columna* amb cost $\Theta(1)$.

Problema 4

(3 punts)

Assumiu que n és una potència de 2, és a dir, de la forma 2^k per a un cert $k \geq 0$.

Una matriu quadrada A de nombres reals de mida $n \times n$ és una *matriu de Monge* si per qualssevol i, j, k i l tals que $0 \leq i < k < n$ i $0 \leq j < l < n$, es compleix que

$$A_{i,j} + A_{k,l} \leq A_{i,l} + A_{k,j}$$

En altres paraules, sempre que agafem dues files i dues columnes d'una matriu de Monge i considerem els quatre elements a les interseccions de les files i les columnes, la suma dels elements de les cantonades superior esquerra i inferior dreta és més petita o igual que la suma dels elements de les cantonades superior dreta i inferior esquerra. Per exemple, la següent matriu és de Monge:

$$\begin{pmatrix} 10 & 17 & 13 & 28 \\ 16 & 22 & 16 & 29 \\ 24 & 28 & 22 & 34 \\ 11 & 13 & 6 & 6 \end{pmatrix}$$

- (a) (1 punt) Sigui $f_A(i)$ l'índex de la columna on apareix l'element mínim de la fila i -èsima (desempatant si cal agafant el de la columna de més a l'esquerra). Per exemple, a la matriu de dalt $f_A(0) = 0$, $f_A(1) = 0$, $f_A(2) = 2$ i $f_A(3) = 2$.

Demostreu que $f_A(0) \leq f_A(1) \leq \dots \leq f_A(n-1)$ per qualsevol matriu de Monge A de mida $n \times n$.

(b) (1 punt) A continuació es descriu un algorisme de dividir i vèncer que calcula la funció f_A per a totes les files d'una matriu de Monge A :

- (1) Construïm dues submatrius quadrades B_1 i B_2 de la matriu A de mida $\frac{n}{2} \times \frac{n}{2}$ de la forma següent: B_1 està formada per les files d' A amb índex parell i les columnes entre 0 i $\frac{n}{2} - 1$, i B_2 està formada per les files d' A amb índex parell i les columnes entre $\frac{n}{2}$ i $n - 1$.
- (2) Recursivament determinem la columna on apareix el mínim més a l'esquerra per a tota fila de B_1 i de B_2 .
- (3) Calculem la columna on apareix el mínim més a l'esquerra de tota fila d' A .

Expliqueu com, a partir del resultat de (2), es pot fer (3) en temps $\Theta(n)$.

- (c) (1 punt) Calculeu el cost en funció de n de l'algorisme proposat a l'apartat anterior per calcular la funció f_A per a totes les files d'una matriu de Monge A de mida $n \times n$. Assumiu que el pas (1) es pot dur a terme en temps $\Theta(n)$.

Examen Parcial EDA

Duració: 2.5 hores

07/11/2016

Problema 1

(2 punts)

En aquest problema no cal justificar les respostes.

- (a) (0.8 pts.) Ompliu els buits de la taula següent (excepte la casella marcada amb **No ompliu**) amb els costos en temps per ordenar un vector d'enters de mida n usant els algorismes que s'indiquen. Assumiu probabilitat uniforme en el cas mig.

	<i>Cas millor</i>	<i>Cas mig</i>	<i>Cas pitjor</i>
Quicksort (amb partició de Hoare)			
Mergesort			
Inserció		No ompliu	

- (b) (0.2 pts.) La solució de la recurrència $T(n) = 2T(n/4) + \Theta(1)$ és

- (c) (0.2 pts.) La solució de la recurrència $T(n) = 2T(n/4) + \Theta(\sqrt{n})$ és

- (d) (0.2 pts.) La solució de la recurrència $T(n) = 2T(n/4) + \Theta(n)$ és

- (e) (0.3 pts.) Què calcula l'algorisme de Karatsuba? Quin cost té?

- (f) (0.3 pts.) Què calcula l'algorisme de Strassen? Quin cost té?

Problema 2**(3 punts)**

Donat un $n \geq 2$, diem que una seqüència de n enters a_0, \dots, a_{n-1} és *bicreixent* si $a_{n-1} < a_0$ i existeix un índex t (amb $0 \leq t < n$) que satisfà les següents condicions:

- $a_0 \leq \dots \leq a_{t-1} \leq a_t$
- $a_{t+1} \leq a_{t+2} \leq \dots \leq a_{n-1}$

Per exemple, la seqüència 12,12,15,20,1,3,3,5,9 és bicreixent (preneu $t = 3$).

(a) (2 pts.) Implementeu en C++ una funció

```
bool search(const vector<int>& a, int x);
```

que, donats un vector a que conté una seqüència bicreixent i un enter x , retorni si x apareix a la seqüència o no. Si useu funcions auxiliars que no siguin de la llibreria estàndard de C++, implementeu-les també. Cal que la solució tingui cost $\Theta(\log(n))$ en temps en el cas pitjor.

(b) (1 pt.) Justifiqueu que el cost en temps en el cas pitjor de la vostra funció *search* és $\Theta(\log(n))$. Quan es dona aquest cas pitjor?

Problema 3

(2 punts)

Considereu la següent funció:

```
int mystery(int m, int n) {  
    int p = 1;  
    int x = m;  
    int y = n;  
    while (y != 0) {  
        if (y % 2 == 0) {  
            x *= x;  
            y /= 2;  
        }  
        else {  
            y -= 1;  
            p *= x;  
        }  
    }  
    return p;  
}
```

- (a) (1 pt.) Donats dos enters $m, n \geq 0$, què calcula $mystery(m, n)$? No cal justificar la resposta.

(b) (1 pt.) Analitzeu el cost en temps en el cas pitjor en funció de n de $mystery(m, n)$.

Problema 4**(3 punts)**

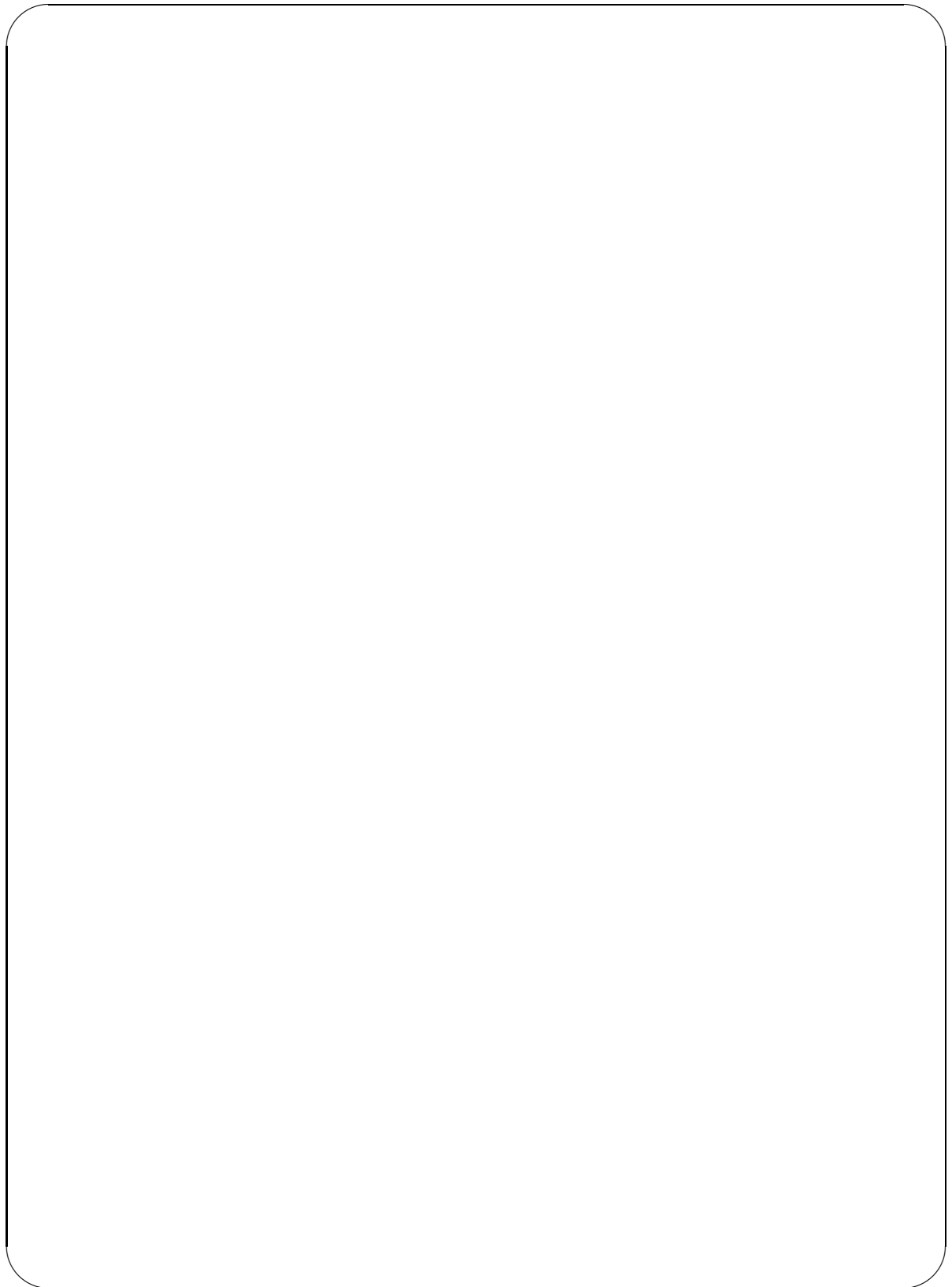
La successió de Fibonacci ve definida per la recurrència $F(0) = 0$, $F(1) = 1$ i

$$F(n) = F(n-1) + F(n-2) \quad \text{si } n \geq 2.$$

(a) (0.5 pts.) Sigui $\phi = \frac{\sqrt{5}+1}{2}$ l'anomenat *nombre d'or*. Demostreu que $\phi^{-1} = \phi - 1$.

(b) (1.5 pts.) Demostreu que per a tot $n \geq 0$ es té que

$$F(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}$$



(c) (1 pt.) Demostreu que $F(n) = \Theta(\phi^n)$.



Examen Parcial EDA

Duració: 2.5 hores

20/04/2017

Problema 1

(2 punts)

- (a) (0.5 pts.) Calculeu el cost mig de l'algorisme d'ordenació per inserció per ordenar un vector de n enters diferents quan amb probabilitat $\frac{\log n}{n}$ s'escull un vector ordenat del revés i amb probabilitat $1 - \frac{\log n}{n}$ s'escull un vector ordenat.

- (b) (0.5 pts.) Considereu la següent funció:

```
bool mystery(int n) {  
    if (n ≤ 1) return false;  
    if (n == 2) return true;  
    if (n%2 == 0) return false;  
    for (int i = 3; i*i ≤ n; i += 2)  
        if (n%i == 0)  
            return false;  
    return true;  
}
```

Completeu: la funció *mystery* calcula i el seu cost és $O(\text{ })$. No cal justificar la resposta.

- (c) (0.5 pts.) Demostreu que $n! = O(n^n)$ aplicant la definició (sense usar límits).

(d) (0.5 pts.) Demostreu que $n! = \Omega(2^n)$ aplicant la definició (sense usar límits).

Problema 2

(3 punts)

Donat un $n \geq 1$, es diu que una seqüència de n enters a_0, \dots, a_{n-1} és *unimodal* si existeix t amb $0 \leq t < n$ tal que $a_0 < \dots < a_{t-1} < a_t$ i $a_t > a_{t+1} > \dots > a_{n-1}$. A l'element a_t se l'anomena el *cim* de la seqüència.

Per exemple, la seqüència 1,3,5,9,4,1 és unimodal, i el seu cim és 9 (preneu $t = 3$).

(a) (1.5 pts.) Implementeu una funció `int top(const vector<int>& a)` en C++ que, donat un vector no buit a que conté una seqüència unimodal, retorni l'índex del cim de la seqüència. Si useu funcions auxiliars que no siguin de la llibreria estàndard de C++, implementeu-les també. Cal que la solució tingui cost $\Theta(\log n)$ en temps en el cas pitjor. Justifiqueu que el cost és en efecte $\Theta(\log n)$, i doneu una situació en què es pot donar aquest cas pitjor.

- (b) (1.5 pts.) Implementeu una funció `bool search(const vector<int>& a, int x)` en C++ que, donat un vector no buit a que conté una seqüència unimodal i un enter x , retorni si x apareix a la seqüència o no. Si useu funcions auxiliars que no siguin de la llibreria estàndard de C++, implementeu-les també. Cal que la solució tingui cost $\Theta(\log n)$ en temps en el cas pitjor. Justifiqueu que el cost és en efecte $\Theta(\log n)$, i doneu una situació en què es pot donar aquest cas pitjor.

Problema 3**(2 punts)**

La següent classe *vect* és una simplificació de la classe **vector** de la STL:

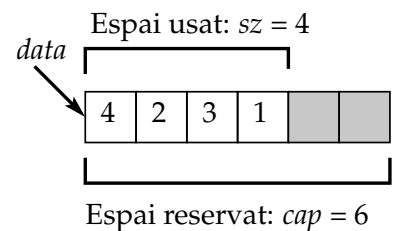
```
class vect {  
    int sz, *data, cap;  
    int new_capacity ();  
    void reserve (int new_cap);  
public:
```

```

vect ()                { sz = cap = 0; data = nullptr; }
int& operator[] (int index) { return data[index]; }
int size ()            { return sz; }
void push_back (int value) {
    if (sz ≥ cap) reserve (new_capacity ());
    data[sz] = value;
    ++sz;
}

```

En un objecte de la classe *vect*, el camp *sz* guarda el nombre d'elements que actualment el vector conté. La memòria reservada per al vector, apuntada pel camp *data*, sempre té espai per guardar els *sz* elements actuals, i possiblement algun més. Al nombre màxim d'elements que es podrien guardar en la memòria reservada se l'anomena capacitat, i és el valor del camp *cap*. El diagrama de la dreta il·lustra una possible implementació d'un vector amb contingut (4,2,3,1). Noteu que les caselles grises estan reservades però no s'usen.



La raó d'aquesta estructura de dades és que cridar al sistema per demanar memòria és una operació costosa que no es vol fer sovint. La funció **void** *reserve* (**int** *new_cap*), la implementació de la qual no es detalla, s'encarrega d'això: demana un nou fragment de memòria prou gran com per poder-hi encabir *new_cap* elements, hi copia el contingut del vector antic i actualitza convenientment els camps *sz*, *data* i *cap*.

Observeu que, cada vegada que es crida la funció *push_back*, si no hi ha prou espai en reserva, es crida la funció **int** *new_capacity* () que, a partir de la capacitat actual, determina la nova capacitat per a la funció *reserve*.

(a) (0.5 pts.) Considereu la següent implementació de la funció *new_capacity*:

```

int new_capacity () {
    const int A; // A és un paràmetre prefixat tal que A ≥ 1
    return cap + A; }

```

Quant val exactament *cap* en un vector que inicialment té *cap* = 0 i al qual s'ha aplicat l'operació *reserve* *m* cops? Anomenem $C(m)$ a aquesta quantitat.

Si fem *n* crides a *push_back* sobre un vector inicialment buit (*sz* = *cap* = 0), quants cops **exactament** s'ha hagut de fer la crida a *reserve* (és a dir, quin és el valor de *m* tal que $n \leq C(m)$ i $C(m-1) < n$)? Doneu-ne també una expressió asimptòtica el més precisa i simple possible.

- (b) (0.75 pts.) Demostreu que la solució de la recurrència definida per $C(0) = 0$, $C(m+1) = AC(m) + B$, on $A > 1$ i $B \geq 1$, és $C(m) = \frac{BA^m - B}{A-1}$ per a $m \geq 0$.

- (c) (0.75 pts.) El mateix que a l'apartat (a), però amb la següent funció *new_capacity*:

```
int new_capacity() {  
    const int A, B; // A, B són paràmetres prefixats tals que  $A > 1$ ,  $B \geq 1$   
    return A*cap + B; }
```

Problema 4**(3 punts)**

Volem tenir una funció

`int stable_partition (int x, vector<int>& a)`

que, donats un enter x i un vector de n enters diferents $a = (a_0, a_1, \dots, a_{n-1})$, reordena el contingut del vector de forma que tots els elements del subvector $a[0..k]$ són menors o iguals a x , i tots els elements del subvector $a[k + 1..n - 1]$ són majors que x , i també torna l'índex k ($k = -1$ si tots els elements de a són majors que x).

A més, l'ordre relatiu original dels elements de a es respecta:

- si $a[i]$ i $a[j]$ amb $i < j$ són tots dos menors o iguals que x , llavors en el vector final $a[i]$ estarà abans que $a[j]$, i tots dos s'hauran col·locat a la "part" $a[0..k]$ que conté els elements $\leq x$;
- si $a[i]$ i $a[j]$ amb $i < j$ són tots dos majors que x , llavors en el vector final $a[i]$ estarà abans que $a[j]$, i tots dos s'hauran col·locat a la "part" $a[k + 1..n - 1]$ que conté els elements $> x$.

Per exemple, donats $x = 2$ i la seqüència $a = (1, 5, 3, 0, 4)$, voldríem que la funció actualitzés a a $(1, 0, 5, 3, 4)$, i que retornés $k = 1$.

- (a) (1 pt.) Implementeu la funció `stable_partition` en C++. Cal que el cost en temps sigui $\Theta(n)$. Justifiqueu el cost. Quin és el cost en espai de la memòria auxiliar que fa servir la vostra implementació?

- (b) (0.5 pts.) Què fa la següent funció *mystery*? No cal justificar la resposta.

```

void mystery_aux(vector<int>& a, int l, int r) {
    // Pre:  $0 \leq l \leq r < a.size()$ 
    for (int i = l, j = r; i < j; ++i, --j) swap(a[i], a[j]);
}

void mystery(vector<int>& a, int l, int p, int r) {
    // Pre:  $0 \leq l \leq p \leq r < a.size()$ 
    mystery_aux(a, l, p);
    mystery_aux(a, p+1, r);
    mystery_aux(a, l, r);
}

```

- (c) (1.5 pts.) Ompliu els buits de la següent implementació alternativa de *stable_partition* i analitzeu-ne el cost en temps en el cas pitjor. Assumiu que, en el cas pitjor, a cada crida recursiva de *stable_partition_rec* es té que $q - p = \Theta(r - l + 1)$.

```

int stable_partition (int x, vector<int>& a) {
    return stable_partition_rec (x, a, 0, a.size()-1);
}

int stable_partition_rec (int x, vector<int>& a, int l, int r) {
    if (l == r) {
        if (a[l] ≤ x) return l;
        else return  ;
    }
    int m = (l+r)/2;
    int p = stable_partition_rec (x, a, l, m);
    int q = stable_partition_rec (x, a, , r);
    mystery(a, , m, );
    return  ; }

```

Anàlisi del cost:

Examen Parcial EDA

Duració: 2.5 hores

06/11/2017

Problema 1

(3 punts)

Responen les següents preguntes. No cal justificar les respostes.

(a) (0.5 pts.) El cost en temps del següent fragment de codi en funció de n :

```
int j = 0;
int s = 0;
for (int i = 0; i < n; ++i)
    if (i == j*j) {
        for (int k = 0; k < n; ++k) ++s;
        ++j;
    }
```

és $\Theta(\quad)$.

(b) (1 pt.) Donats un vector d'enters v i un enter x , la funció

```
int posicio (const vector<int>& v, int x) {
    int n = v.size ();
    for (int i = 0; i < n; ++i)
        if (v[i] == x)
            return i;
    return -1;
}
```

examina les $n = v.size()$ posicions de v i retorna la primera que conté x , o -1 si no n'hi ha cap.

Suposem que x apareix al vector v . El cost asimptòtic en temps de *posicio* en el cas pitjor és $\Theta(\quad)$, i en el cas mig (assumint probabilitat uniforme) és $\Theta(\quad)$.

(c) (0.5 pts.) Doneu l'ordre de magnitud de la següent funció de la forma més senzilla possible: $5(\log n)^2 + 2\sqrt{n} + \cos(n^8) = \Theta(\quad)$.

(d) (0.5 pts.) La solució de la recurrència

$$T(n) = \begin{cases} 1 & \text{si } 0 \leq n < 2 \\ 4 \cdot T(n/2) + 3n^2 + 2\log(\log n) + 1, & \text{si } n \geq 2 \end{cases}$$

és $T(n) = \Theta(\quad)$.

(e) (0.5 pts.) La solució de la recurrència

$$T(n) = \begin{cases} 1 & \text{si } 0 \leq n < 2 \\ 4 \cdot T(n-2) + 3n^2 + 2\log(\log n) + 1, & \text{si } n \geq 2 \end{cases}$$

és $T(n) = \Theta(\quad)$.

Problema 2

(2.5 punts)

Considereu el següent programa, que llegeix un enter estrictament positiu m i una seqüència de n enters a_0, a_1, \dots, a_{n-1} que es garanteix que es troben entre 1 i m :

```
int main() {
    int m;
    cin >> m;
    vector<int> a;
    int x;
    while (cin >> x)
        a.push_back(x);

    vector<int> b(m + 1, 0);
    int n = a.size();
    for (int i = 0; i < n; ++i)
        ++b[a[i]];

    for (int j = 1; j <= m; ++j)
        b[j] += b[j-1];
}
```

(a) (0.5 pts.) Donat un y tal que $0 \leq y \leq m$, quant val $b[y]$ al final del *main* en termes de la seqüència a ?

(b) (1 pt.) Definim la *mediana* de la seqüència a com el valor p entre 1 i m tal que $b[p] \geq \frac{n}{2}$ i $b[p-1] < \frac{n}{2}$, on b és el vector calculat com en el codi mostrat a dalt.

Escriviu en C++ una funció

```
int median(int n, const vector<int>& b);
```

que, a partir de n , la mida de la seqüència a , i de b , calculat com a dalt, trobi la mediana de a . No es consideraran com a respostes vàlides les solucions amb cost $\Omega(m)$. Si feu servir funcions auxiliars, implementeu-les també.

(c) (1 pt.) Analitzeu el cost de la vostra funció *median* de l'apartat anterior en funció de m .

Problema 3**(2 punts)**

Considereu el tipus

```
struct complex {  
    int real ;  
    int imag;  
};
```

per a implementar nombres complexos (amb components enteres), on *real* és la part real i *imag* és la part imaginària del nombre complex.

Per exemple, si z és un objecte de tipus *complex* que representa el nombre $2 + 3i$, llavors $z.\text{real} = 2$ i $z.\text{imag} = 3$.

(a) (1 pt.) Implementeu en C++ una funció

```
complex exp(complex z, int n);
```

que, donat un nombre complex z i un enter $n \geq 0$, calculi el nombre complex z^n . Cal que la solució tingui cost $\Theta(\log n)$ en temps. Si feu servir funcions auxiliars, implementeu-les també.

Nota. Recordeu que el producte de nombres complexos es defineix així:

$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$$

(b) (1 pt.) Justifiqueu que el cost en temps de la vostra funció *exp* és $\Theta(\log n)$.

Problema 4**(2.5 punts)**

Considereu la següent recurrència:

$$T(n) = T(n/2) + \log n$$

(a) (1 pt.) Definim la funció U com $U(m) = T(2^m)$. A partir de la recurrència de T , deduiu una recurrència per a U .

- (b) (0.5 pts.) Resoleu asimptòticament la recurrència per a U de la resposta de l'apartat anterior.

- (c) (1 pt.) Resoleu asimptòticament la recurrència per a T .

Examen Parcial EDA

Duració: 2.5 hores

19/04/2018

Problema 1

(1.5 punts)

(a) (0.5 pts.) Considereu les funcions $f(n) = n^{n^2}$ i $g(n) = 2^{2^n}$.

Quina funció de les dues creix asimptòticament més de pressa?

Justificació:

Nota: Cal entendre n^{n^2} com $n^{(n^2)}$, i similarmet 2^{2^n} com $2^{(2^n)}$.

(b) (0.5 pts.) Supposeu que les entrades de mida n d'un cert algorisme són dels tipus següents:

- Tipus 1: per cada entrada d'aquest tipus, l'algorisme triga temps $\Theta(n^4)$. A més, la probabilitat que l'entrada sigui d'aquest tipus és $\frac{1}{n^3}$.
- Tipus 2: per cada entrada d'aquest tipus, l'algorisme triga temps $\Theta(n^3)$. A més, la probabilitat que l'entrada sigui d'aquest tipus és $\frac{1}{n}$.
- Tipus 3: per cada entrada d'aquest tipus, l'algorisme triga temps $\Theta(n)$. A més, la probabilitat que l'entrada sigui d'aquest tipus és $1 - \frac{1}{n^3} - \frac{1}{n}$.

Aleshores el cost de l'algorisme en el cas mig és .

Justificació:

(c) (0.5 pts.) Resoleu la recurrència $T(n) = 2T(n/4) + \Theta(\sqrt{n})$.

Resposta: $T(n) = \Theta(\text{ })$.

Justificació:

Problema 2**(2 punts)**

En aquest problema prendrem n com a una constant. Donats un vector $x_0 \in \mathbb{Z}^n$ i una matriu quadrada $A \in \mathbb{Z}^{n \times n}$, definim la successió $x(0), x(1), x(2), \dots$ així:

$$x(k) = \begin{cases} x_0 & \text{si } k = 0 \\ A \cdot x(k-1) & \text{si } k > 0 \end{cases}$$

Per exemple, per a $n = 3$, si

$$x_0 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \text{i} \quad A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix},$$

tenim que

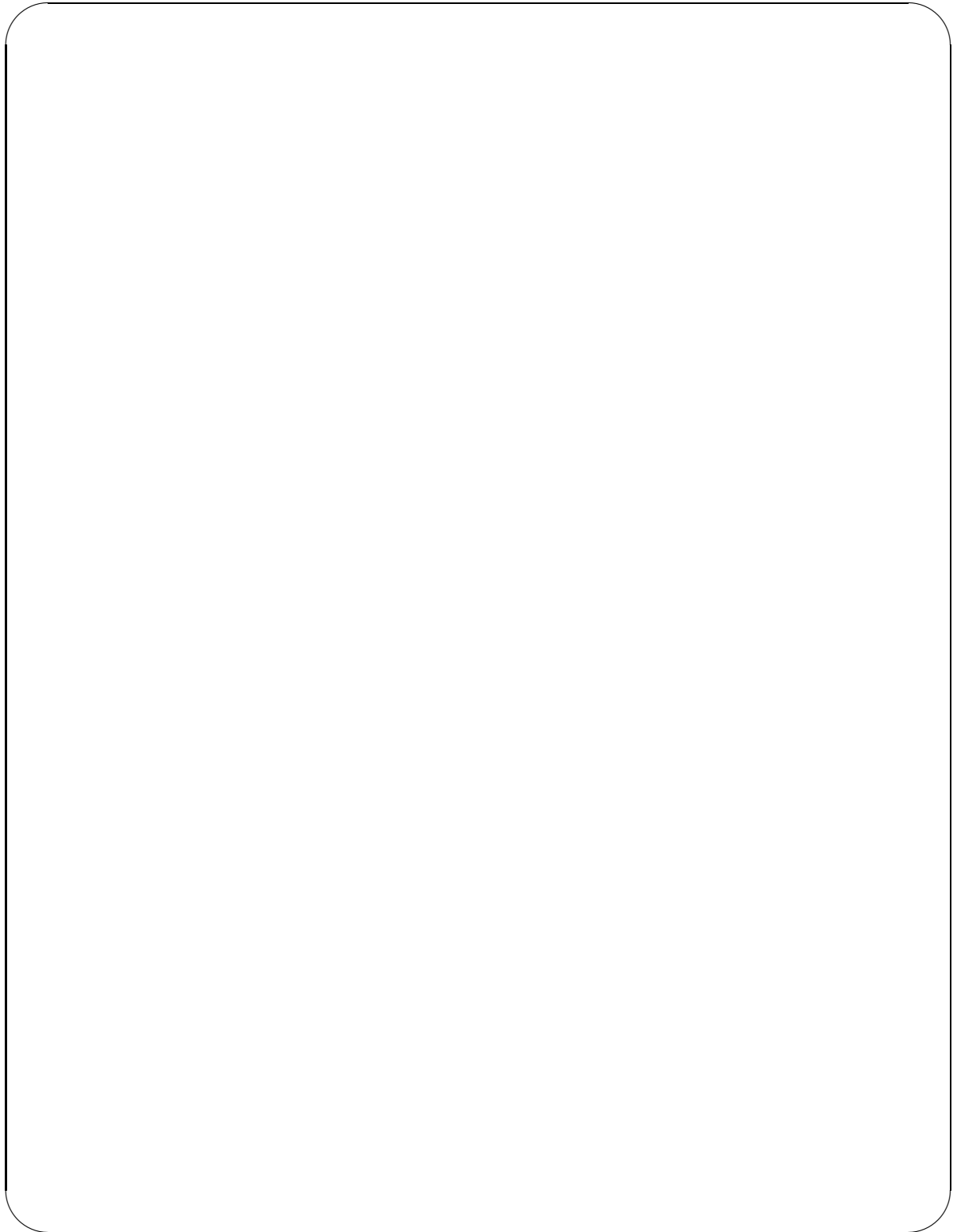
$$x(0) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad x(1) = \begin{pmatrix} 6 \\ 1 \\ 2 \end{pmatrix}, \quad x(2) = \begin{pmatrix} 9 \\ 6 \\ 1 \end{pmatrix}, \quad \dots$$

(a) (1 pt.) Demostreu per inducció que $x(k) = A^k \cdot x_0$ per a tot $k \geq 0$.

(b) (1 pt.) Expliqueu a alt nivell com implementaríeu una funció

vector<int> k_th(const vector<vector<int>>& A, const vector<int>& x0, int k);

per calcular el terme k -èsim $x(k)$ de la successió definida per la matriu A i el vector x_0 . Analitzeu-ne també el cost en temps en funció de k . Cal que el cost sigui millor que $\Theta(k)$.

**Problema 4****(3 punts)**

- (a) (1 pt.) Considereu una recurrència de la forma

$$T(n) = T(n/b) + \Theta(\log^k n)$$

on $b > 1$ i $k \geq 0$. Demostreu que la seva solució és $T(n) = \Theta(\log^{k+1} n)$.

Justificació:

Nota: $\log^k n$ és una forma breu d'escriure $(\log(n))^k$.

Pista: feu un canvi de variable.

- (b) (1 pt.) Donat un $n \geq 1$, es diu que una seqüència de n enters a_0, \dots, a_{n-1} és *unimodal* si existeix t amb $0 \leq t < n$ tal que $a_0 < \dots < a_{t-1} < a_t$ i $a_t > a_{t+1} > \dots > a_{n-1}$. A l'element a_t se l'anomena el *cim* de la seqüència. Per exemple, la seqüència 1,3,5,9,4,1 és unimodal, i el seu cim és 9 (preneu $t = 3$).

Ompliu els buits del codi a continuació per tal que la funció

```
bool search(const vector<int>& a, int x),
```

donat un vector no buit a que conté una seqüència unimodal i un enter x , retorni si x apareix a la seqüència o no. No cal justificació.

```
bool search(const vector<int>& a, int x, int l, int r) {
    if (  ) return x == a[l];
    int m = (l+r)/2;
    auto beg = a.begin();
    if (a[m] < )
        return search(a, x, m+1, r) or binary_search(beg + l, beg + , x);
    else
        return search(a, x, l, m) or binary_search(beg + , beg + r + 1, x);
}
```

```
bool search(const vector<int>& a, int x) { return search(a, x, 0,  ); }
```

Nota: Donats un element x i iteradors $first, last$ tals que l'interval $[first, last)$ està ordenat (creixentment o decreixentment), la funció $binary_search(first, last, x)$ retorna **true** si x apareix a l'interval $[first, last)$ i **false** altrament, en temps logarítmic en la mida de l'interval en el cas pitjor.

- (c) (1 pt.) Analitzeu el cost en temps en el cas pitjor d'una crida $search(a, x)$, on a és un vector de mida $n > 0$ que conté una seqüència unimodal i x és un enter. Descriviu una situació en què es pugui donar aquest cas pitjor.

Examen Parcial EDA

Duració: 2.5 hores

05/11/2018

Problema 1

(2 punts)

Per cada afirmació donada a continuació, marqueu amb una X la casella corresponent segons si és certa o falsa.

Nota: Cada resposta correcta sumarà 0.2 punts; cada resposta equivocada restarà 0.2 punts, llevat del cas que hi hagi més respostes equivocades que correctes, en què la nota de l'exercici serà 0.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
CERT										
FALS										

- (1) El cost en temps de quicksort en el cas pitjor és $\Theta(n^2)$.
- (2) El cost en temps de quicksort en el cas mig és $\Theta(n^2)$.
- (3) Qualsevol algorisme que calculi la suma $x + y$ de dos naturals x, y de n bits cadascun, ha de tenir cost en temps $\Omega(n)$.
- (4) Existeix un algorisme que calcula la suma $x + y$ de dos naturals x, y de n bits cadascun, en temps $O(n)$.
- (5) Qualsevol algorisme que calculi el producte $x \cdot y$ de dos naturals x, y de n bits cadascun, ha de tenir cost en temps $\Omega(n^2)$.
- (6) Existeix un algorisme que calcula el producte $x \cdot y$ de dos naturals x, y de n bits cadascun, en temps $O(n^2)$.
- (7) Qualsevol algorisme que, donats un enter $k \geq 0$ i una matriu A de $n \times n$ nombres enters, calculi la matriu A^k ha de fer $\Omega(k)$ productes de matrius.
- (8) $2^{2n} \in O(2^n)$.
- (9) $2n \in O(n)$.
- (10) $\log(2n) \in O(\log n)$.

Problema 2

(3 punts)

Donades dues matrius de booleans A i B de mida $n \times n$, definim el seu *producte lògic* $P = A \cdot B$ com la matriu $n \times n$ que al coeficient de la fila i -èsima i columna j -èsima ($0 \leq i, j < n$) conté:

$$p_{ij} = \bigvee_{k=0}^{n-1} (a_{ik} \wedge b_{kj})$$

- (a) (0.5 pts.) Considereu la següent funció per a calcular el producte lògic:

```

vector<vector<bool>> product1(const vector<vector<bool>>& A,
                             const vector<vector<bool>>& B) {
    int n = A.size ();
    vector<vector<bool>> P(n, vector<bool>(n, false));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                P[i][j] = P[i][j] or (A[i][k] and B[k][j]);
    return P;
}

```

Quin és el cost en temps en el cas pitjor en termes de n ? El cost és $\Theta(\text{ })$.

Doneu un exemple de cas pitjor.

(b) (1 pt.) Considereu la següent alternativa per a calcular el producte lògic:

```

vector<vector<bool>> product2(const vector<vector<bool>>& A,
                             const vector<vector<bool>>& B) {
    int n = A.size ();
    vector<vector<bool>> P(n, vector<bool>(n, false));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n and not P[i][j]; ++k)
                if (A[i][k] and B[k][j]) P[i][j] = true;
    return P;
}

```

Quin és el cost en temps en el cas millor en termes de n ? El cost és $\Theta(\text{ })$.

Doneu un exemple de cas millor.

Quin és el cost en temps en el cas pitjor en termes de n ? El cost és $\Theta(\text{ })$.

Doneu un exemple de cas pitjor.

- (c) (1.5 pt.) Expliqueu a alt nivell com implementaríeu una funció per a calcular el producte lògic de matrius que fos més eficient asimptòticament en el cas pitjor que les proposades als apartats (a) i (b). Quin és el seu cost en temps en el cas pitjor?

Problema 3**(2 punts)**

Considereu la següent funció:

```
int mystery_rec(int n, int l, int r) {  
    if (r == l+1) return l;  
    int m = (l+r)/2;  
    if (m*m ≤ n) return mystery_rec(n, m, r);  
    else return mystery_rec(n, l, m);  
}
```

```
int mystery(int n) {  
    return mystery_rec(n, 0, n+1);  
}
```

- (a) (1 pt.) Donat un enter $n \geq 0$, què calcula la funció *mystery*?

- (b) (1 pt.) Quin cost en temps té *mystery*(n) en funció de n ? El cost és $\Theta(\text{ })$

Raoneu la resposta.

Problema 4

(3 punts)

Considerem el problema de, donat un vector d'enters (possiblement amb repeticions), ordenar-lo de forma creixent. Els costos a continuació són en temps i es demanen en funció de n , la mida del vector.

- (a) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per inserció en el cas millor? El cost és $\Theta(\text{ })$.
- (b) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per inserció en el cas pitjor? El cost és $\Theta(\text{ })$.

- (c) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per fusió en el cas millor? El cost és $\Theta(\text{ })$.
- (d) (0.25 pts.) Quin és el cost de l'algorisme d'ordenació per fusió en el cas pitjor? El cost és $\Theta(\text{ })$.
- (e) (0.75 pts.) Ompliu els buits de la funció *my_sort* definida a continuació, per tal que donat un vector d'enters *v*, l'ordini de forma creixent:

```

void my_sort(vector<int>& v) {
    int n = v.size ();
    double lim = n * log(n);
    int c = 0;
    for (int i = 1; i < n; ++i) {
        int x = v[i];
        int j;
        for (j = i; j > 0 and  ; --j) {
            v[j] =  ;
            ++c;
        }
        v[j] =  ;
        if (c > lim) {
            merge_sort(v);
            return;
        }
    }
}

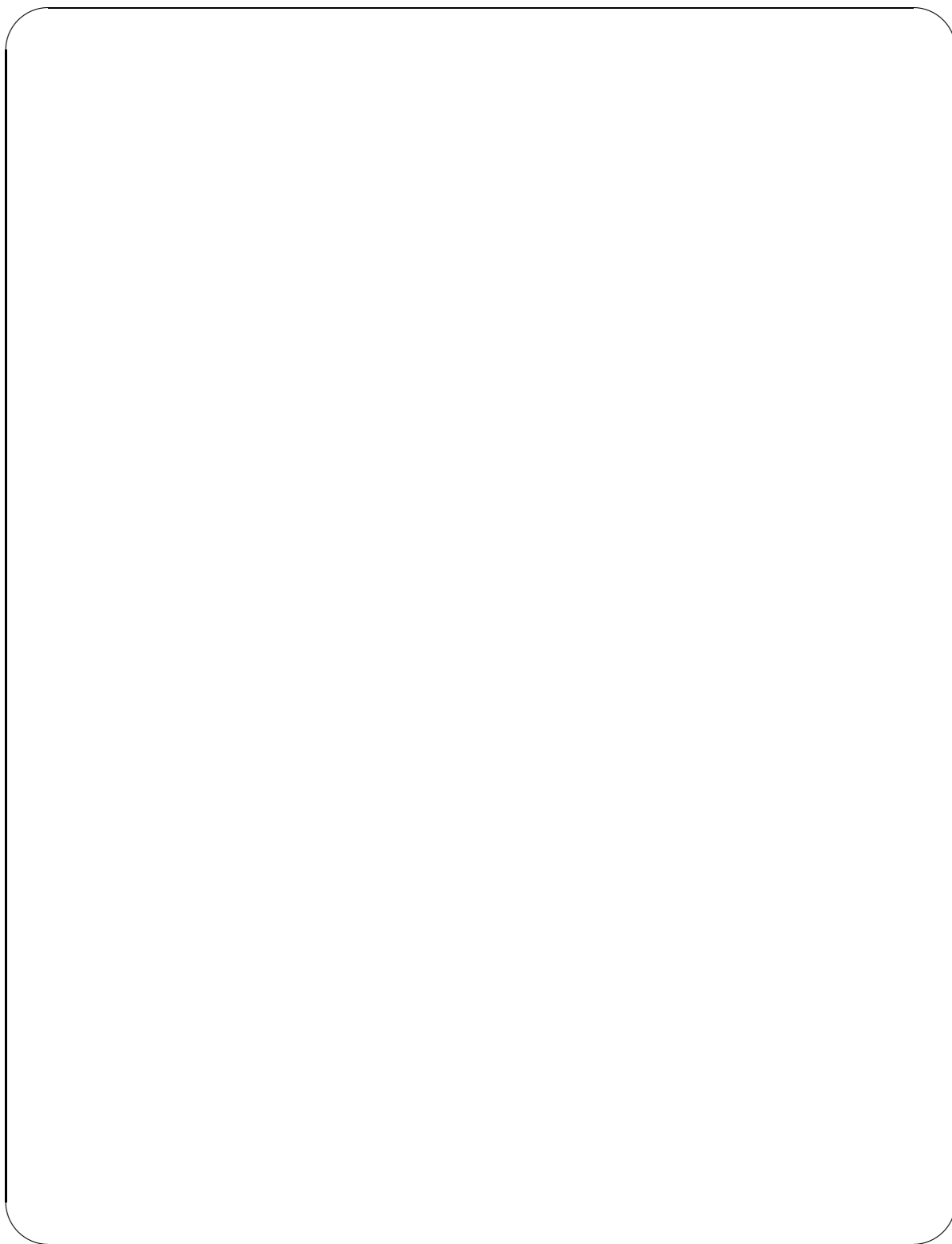
```

La funció auxiliar *merge_sort* és una implementació de l'algorisme d'ordenació per fusió, i la funció *log* calcula el logaritme neperià (això és, en base *e*).

- (f) (1.25 pts.) Quin és el cost de *my_sort* en el cas millor? El cost és $\Theta(\text{ })$.

Quin és el cost de *my_sort* en el cas pitjor? El cost és $\Theta(\text{ })$.

Raoneu les vostres respostes i doneu un exemple de cas millor i un de cas pitjor.



Examen Parcial EDA

Duració: 2.5 hores

25/04/2019

Problema 1

(3 punts)

En aquest problema no cal justificar les respostes.

- (a) (1 pt.) Considereu les funcions $f(n) = n \log n$ i $g(n) = n^\alpha$, on α és un paràmetre real. Determineu per a quins valors d' α :

(1) $f = O(g)$:

(2) $f = \Omega(g)$:

(3) $g = O(f)$:

(4) $g = \Omega(f)$:

- (b) (0.5 pts.) La solució de la recurrència $T(n) = 2T(n-2) + \Theta(n)$ és asimptòticament $T(n) = \Theta(\text{ })$.

- (c) (0.5 pts.) La solució de la recurrència $T(n) = 2T(n/2) + \Theta(n)$ és asimptòticament $T(n) = \Theta(\text{ })$.

- (d) (0.5 pts.) El cost en el cas pitjor de mergesort per ordenar un vector de mida n en funció de n és $\Theta(\text{ })$.

- (e) (0.5 pts.) El cost en el cas millor de mergesort per ordenar un vector de mida n en funció de n és $\Theta(\text{ })$.

Problema 2

(2 punts)

Considereu el programa següent:

```
int partition (vector<int>& v, int l, int r) {
    int x = v[l];
    int i = l - 1;
    int j = r + 1;
    while (true) {
        while (x < v[--j]);
        while (v[++i] < x);
        if (i ≥ j) return j;
        swap(v[i], v[j]);
    }
}

void mystery(vector<int>& v, int l, int r, int m) {
    if (l < r) {
        int q = partition(v, l, r);
        mystery(v, l, q, m);
        int p = q - l + 1;
        if (p < m)
            mystery(v, q + 1, r, m - p);
    }
}
```

- (a) (1 pt.) Donats un vector v i un enter $m \geq 0$, expliqueu què fa una crida $mystery(v, 0, v.size() - 1, m)$.

- (b) (1 pt.) Suposem que v és un vector d'enters diferents ordenats de forma creixent. Sigui $n = v.size()$. Analitzeu el cost de cridar $mystery(v, 0, n - 1, n)$ en funció de n .

Problema 3**(2 punts)**

En aquest problema volem multiplicar nombres naturals arbitràriament grans. Un natural s'implementa amb un `vector<bool>` usant la seva representació en bits, de més a menys significatiu. Així, per exemple, el vector (1, 1, 0) representa el nombre $4 + 2 = 6$.

Suposem que ja hem implementat les funcions següents, amb els costos indicats:

```
// Calcula  $x \times 2$ .  
// Cost:  $\Theta(n)$ , on  $n = x.size()$ .  
vector<bool> twice(const vector<bool>& x)  
  
// Calcula  $x \div 2$  (divisió entera per defecte).  
// Cost:  $\Theta(n)$ , on  $n = x.size()$ .  
vector<bool> half(const vector<bool>& x)  
  
// Calcula  $x + y$ .  
// Cost:  $\Theta(n)$ , on  $n = \max(x.size(), y.size())$ .  
vector<bool> sum(const vector<bool>& x, const vector<bool>& y)
```

- (a) (1.5 pts.) Usant aquestes funcions, completeu la implementació de la funció

vector<bool> prod(const vector<bool>& x, const vector<bool>& y)

següent per a calcular el producte de x per y :

vector<bool> prod(const vector<bool>& x, const vector<bool>& y) {

if ($x.size() == 0$ **or** $y.size() == 0$) **return**

vector<bool> z = twice(twice(prod(half(x), half(y))));

vector<bool> one = vector<bool>(1, 1);

if ($x.back() == 0$ **and** $y.back() == 0$) **return**

else if ($x.back() == 1$ **and** $y.back() == 0$) **return**

else if ($y.back() == 1$ **and** $x.back() == 0$) **return**

else {

} }

Nota: Si v és un **vector** de la STL, llavors $v.back()$ és equivalent a $v[v.size() - 1]$.

Si $n = x.size() = y.size()$, justifiqueu que el cost de cridar $prod(x, y)$ en funció de n és $\Theta(n^2)$.

- (b) (0.5 pts.) Doneu el nom d'un conegut algorisme per calcular el producte de dos naturals x i y de n bits cadascun més eficient que la funció *prod* anterior. Quin és el cost d'aquest algorisme en funció de n ? (no cal justificar el cost)

Problema 4

(3 punts)

Una matriu d'enters de dimensions $N \times N$ és *biordenada* si cada columna té els elements ordenats en ordre no decreixent de dalt a baix, i cada fila té els elements ordenats en ordre no decreixent d'esquerra a dreta. Per exemple, una matriu d'aquest tipus seria:

$$\begin{pmatrix} 1 & 4 & 9 \\ 9 & 9 & 9 \\ 12 & 14 & 15 \end{pmatrix}$$

Volem, donats un enter x i una matriu biordenada A , decidir si x apareix a A o no.

- (a) (1.5 pts.) En aquest apartat suposarem que N és una potència de 2. Donats un enter x , una matriu biordenada A , i tres enters i, j, n tals que n és una potència de 2 i que $1 \leq n \leq N$ i $0 \leq i, j \leq N - n$, la funció

bool search1(int x, const vector<vector<int>>& A, int i, int j, int n)

retorna si x apareix a la submatriu $n \times n$ de A que conté de la fila i a la fila $i + n - 1$, i de la columna j a la $j + n - 1$. Files i columnes s'indexen des de 0.

Completeu **usant recursivitat** la implementació següent de *search1*:

```
bool search1(int x, const vector<vector<int>>& A, int i, int j, int n) {
    if (n == 1) return  ;
    int mi = i + n/2 - 1;
    int mj = j + n/2 - 1;

    if (A[mi][mj] < x) return
        

    if (A[mi][mj] > x) return
        

    return  ;
}
```

Nota: a cada caixa es poden fer diverses crides recursives, que podeu combinar amb operadors booleans.

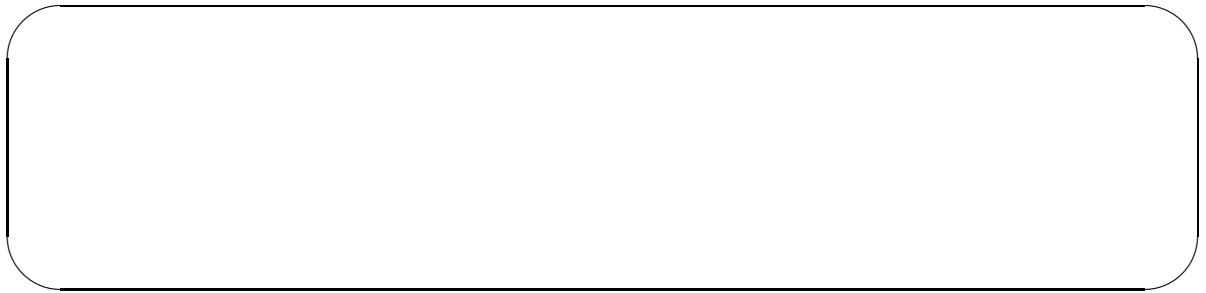
Com es pot utilitzar la funció *search1* per determinar si x apareix a A o no? Analitzeu el cost en el cas pitjor del vostre algorisme per fer-ho en funció de N .

(b) (0.75 pts.) Considereu la funció *search2* següent per dir si x apareix a A o no:

```
bool search2(int x, const vector<vector<int>>& A) {  
    int i = A.size() - 1;  
    int j = 0;  
    while (i ≥ 0 and j < A.size())  
        if (A[i][j] > x) --i;  
        else if (A[i][j] < x) ++j;  
        else return true;  
    return false;  
}
```

Si és correcta, justifiqueu-ho. Si no ho és, doneu-ne un contraexemple.

(c) (0.75 pts.) Analitzeu el cost en el cas pitjor de *search2* en funció de $N = A.size()$.



Examen Parcial EDA

Duració: 2h45min

11/11/2019

Problema 1

(1 punt)

- (a) (0.5 pts.) La solució de la recurrència $T(n) = 2T(n/4) + \Theta(\sqrt{n})$ és asimptòticament $T(n) = \Theta(\text{ })$. No cal que justifiqueu la resposta.
- (b) (0.5 pts.) Per a quines $X \in \{O, \Omega, \Theta\}$ es compleix que $\log_2(n) \in X(\log_2(\log_2(n^2)))$?

Problema 2

(2.5 punts)

Donat un natural $n \geq 1$, qualsevol funció $f: \{0, 1, \dots, n-1\} \rightarrow \{0, 1, \dots, n-1\}$ es pot representar com un vector d'enters $[f(0), f(1), \dots, f(n-1)]$.

Per exemple, si $n = 5$ i $f(0) = 2, f(1) = 1, f(2) = 2, f(3) = 4, f(4) = 3$, la funció f es pot representar pel vector $[2, 1, 2, 4, 3]$. En tot aquest problema, assumirem aquesta representació de funcions.

- (a) (0.75 pts.) Considereu el codi següent:

```
void misteri_aux(const vector<int>& f, const vector<int>& g,
                 int i, vector<int>& r) {
    if (i < f.size()) {
```

```
        r[i] = f[g[i]];
        misteri_aux(f,g,i+1,r);
    }
}

vector<int> misteri(const vector<int>&f, const vector<int>&g) {
    // Precondició: f i g tenen la mateix mida i contenen nombres
    // entre 0 i f.size() - 1
    vector<int> r(f.size());
    misteri_aux(f,g,0,r);
    return r;
}
```

Què retorna la funció *misteri*? No cal que justifiqueu la resposta.

Si assumim que n és la mida de f , quin és el cost de *misteri* en funció de n ?

(b) (0.75 pts.) Considereu ara el codi següent:

```
vector<int> misteri_2(const vector<int>&f, int k) {
    if (k == 0) {
        vector<int> r(f.size());
        for (int i = 0; i < f.size(); ++i) r[i] = i;
        return r;
    }
    else return misteri(f, misteri_2(f,k-1));
}
```

Assumint que $k \geq 0$, què retorna la funció *misteri_2*? No cal que justifiqueu la resposta.

Quin és el cost de *misteri_2* en funció només de *k*?

- (c) (1 pt.) Completeu la funció següent per tal que calculi el mateix que *misteri_2* però sigui més eficient asimptòticament. Analitzeu-ne el cost en funció de *k*.

```
vector<int> misteri_2_quick(const vector<int>& f, int k) {  
    if (k == 0) {  
        vector<int> r(f.size ());  
        for (int i = 0; i < f.size (); ++i) r[i] = i;  
        return r;  
    }
```

}

Anàlisi del cost en funció de k :

Problema 3**(3.25 punts)**

Donat un conjunt S de $m = 2n$ enters diferents, volem agrupar-los en parelles de manera que la suma dels seus productes sigui màxima. És a dir, busquem la màxima expressió de la forma $x_0 * x_1 + x_2 * x_3 + \dots + x_{2n-2} * x_{2n-1}$, on el x_i 's són tots els elements de S .

Per exemple, si $S = \{5, 6, 1, 3, 8, 4\}$, dues possibles expressions són $1 * 5 + 6 * 3 + 4 * 8$, que suma 55, i $5 * 4 + 1 * 8 + 3 * 6$, que suma 46. D'entre aquestes dues preferim la primera, tot i que encara n'hi ha d'altres de millors.

La funció `max_suma` calcula la màxima suma de productes de S :

```
int pos_max (const vector<int>& v, int l, int r) {
    int p = l;
    for (int j = l + 1; j ≤ r; ++j)
        if (v[j] > v[p]) p = j;
    return p;
}
```

```
int max_suma (vector<int>& S) {
    int suma = 0;
    int m = S.size ();
    for (int i = 0; i < m; ++i) {
        int p = pos_max(S, i, m-1);
        swap(S[i], S[p]);
        if (i%2 == 1) suma += S[i-1]*S[i];
    }
    return suma;
}
```

- (a) (1 pt.) Analitzeu el cost en cas pitjor de `max_suma` en funció de m , el nombre d'elements del vector S .

- (b) (1 pt.) Expliqueu a alt nivell com implementaríeu una funció que solucionés el mateix problema però que, en el cas pitjor, fos més eficient asimptòticament que *max_suma*. Indiqueu clarament quin és el cost resultant.

- (c) (1.25 pts.) Demostreu que la funció *max_suma* retorna la suma de productes màxima.

Ajuda: demostreu primer que si x_0 i x_1 son els dos nombres més grans de S , aleshores una expressió que conté els productes $x_0 * y$ i $x_1 * z$, per certs $y, z \in S$ no pot ser màxima. A continuació utilitzeu aquest fet per demostrar la correctesa de *max_suma* per inducció sobre m .

Problema 4

(3.25 punts)

Durant els propers $n \geq 3$ dies, se celebrarà un important esdeveniment esportiu, pel qual existeix un enorme mercat de compra-venda d'entrades del que ens en volem aprofitar. Sabem que cada dia podrem comprar o vendre una entrada, i també sabem el preu de les entrades en cada dia, donat com una seqüència $(p_0, p_1, \dots, p_{n-1})$.

- (a) (1.25 pts.) Ens assabentem que la seqüència de preus segueix una forma ben particular. Hi ha un únic dia $0 \leq d \leq n-1$ amb preu mínim p_d i sabem que $p_0 > p_1 > \dots > p_d$ i $p_d < p_{d+1} < \dots < p_{n-1}$.

El nostre objectiu és comprar una entrada el dia c i vendre-la el dia v , amb $0 \leq c \leq v \leq n-1$ de manera que maximitzem els nostres guanys. És a dir, volem que $p_v - p_c$ sigui màxim. A tal efecte, ompliu els buits del codi següent per tal que la funció *max_guany* retorni aquest parell $\langle c, v \rangle$ en temps $\Theta(\log n)$ i analitzeu per què la funció resultant té aquest cost.

```
int f(const vector<int>& p, int l, int r){
    if (l + 1 ≥ r) return (p[l] ≤ p[r] ? l : r);
    else {
        int m = (l+r)/2;
        if (  ) return f(p, ,  );
        else if (  ) return f(p, ,  );
        else return m;
    }
}

pair<int,int> max_guany (const vector<int>& p) {
    return { ,  };
}
```

Nota: recordeu que l'expressió $(B ? E_T : E_F)$ equival a E_T si l'expressió booleana B és certa i equival a E_F altrament.

Anàlisi del cost:

- (b) (1 pt.) En el que resta d'exercici, assumiu que la seqüència p no necessàriament té la forma mencionada a l'apartat anterior, sinó que és una seqüència arbitrària de nombres naturals.

En aquest apartat, donat un dia k en el que necessitem disposar d'una entrada, volem saber quin és el màxim benefici que podem obtenir comprant l'entrada en un cert dia c i venent-la en un cert dia v , però que ens garanteixi tenir l'entrada el dia k . És a dir, no ens val qualsevol parell (c, v) sinó que necessitem que $0 \leq c \leq k \leq v \leq n - 1$. Implementeu una funció amb cost $\Theta(n)$ per calcular aquest benefici.

```
int max_guany (const vector<int>& p, int k) {
```

```
}
```

- (c) (1 pt.) Afrontem finalment el problema general, en el que la seqüència p pot tenir qual-sevol forma i volem calcular el màxim guany possible $p_v - p_c$ que correspon a comprar una entrada en el dia c i vendre-la posteriorment en el dia v . Expliqueu a alt nivell com implementaríeu una funció que calculés aquest màxim guany i analitzeu-ne el cost. Solucions amb cost $\Omega(n^2)$ rebran 0 punts.

Ajuda: la funció de l'apartat anterior us pot ser útil per implementar una solució basada en dividir i vèncer.

Examen Parcial EDA**Duració: 1h15min****23/04/2020****Problema 1****(3 punt)**

Test https://jutge.org/problems/X71865_ca a omplir a través de Jutge.org. Teniu 20 minuts per entregar les respostes.

Problema 2**(3 punts)**

Un problema escollit aleatòriament d'entre:

Jutge.org, Problema X35804: Creuers
(https://jutge.org/problems/X35804_ca).

Jutge.org, Problema X22314: Donacions
(https://jutge.org/problems/X22314_ca).

Jutge.org, Problema X79163: Efemèrides
(https://jutge.org/problems/X79163_ca).

Problema 3**(4 punts)**

Un problema escollit aleatòriament d'entre:

Jutge.org, Problema X30043: Vector xulo
(https://jutge.org/problems/X30043_ca).

Jutge.org, Problema X74873: Vector xulo
(https://jutge.org/problems/X74873_ca).

Jutge.org, Problema X83303: Vector xulo
(https://jutge.org/problems/X83303_ca).

Jutge.org, Problema X90362: Vector xulo
(https://jutge.org/problems/X90362_ca).

Examen Parcial EDA

Duració: 1h 30min

06/11/2020

Problema 1

(2.5 pts.)

Respon a les següents preguntes:

- (a) (1.5 pts.) Donat un vector v d'enters ordenats creixentment i un enter x , volem determinar si x apareix a v . En lloc d'implementar una cerca binària, ens proposen la idea d'escollir dos elements que parteixin el vector en tres parts iguals i determinar en quina d'aquestes tres parts cal buscar x . Completa el següent codi perquè sigui una implementació correcta d'aquesta idea:

```
bool tri_search (const vector<int>& v, int l, int r, int x) {  
    if (l > r) return false;  
    else {  
        int n_elems = (r-l+1);  
        int f = l + n_elems/3;  
        int s = r - n_elems/3;  
  
        if (  ) return true;  
        if (  ) return tri_search(v,  ,  , x);  
        if (  ) return tri_search(v,  ,  , x);  
        return tri_search (v,  ,  , x);  
    }  
}  
  
bool tri_search (const vector<int>& v, int x) {  
    return tri_search (v, 0, v.size()-1, x);  
}
```

Si n és el nombre d'elements del vector v , analitzeu el cost en cas pitjor d'una crida $tri_search(v, x)$ en funció de n .

- (b) (1 pt.) Doneu dues funcions f i g amb $f \notin \Theta(g)$ tals que ambdues siguin $\Omega(n)$ i $O(n \log n)$ però que cap sigui $\Theta(n)$ ni $\Theta(n \log n)$.

Problema 2

(7.5 pts.)

Donat un vector v d' n naturals volem determinar si existeix un element *dominant*, és a dir, si existeix un element que apareix més de $n/2$ vegades. Per exemple:

- Si $v = \{5, 2, 5, 2, 8, 2, 2\}$, aleshores 2 és l'element dominant perquè apareix $4 > 7/2$ vegades.
- Si $v = \{3, 2, 3, 3, 2, 3\}$, aleshores 3 és l'element dominant perquè apareix $4 > 6/2$ vegades.
- Si $v = \{6, 1, 6, 1, 6, 2, 9\}$, no hi ha cap element dominant perquè cap d'ells apareix més de $7/2$ vegades.

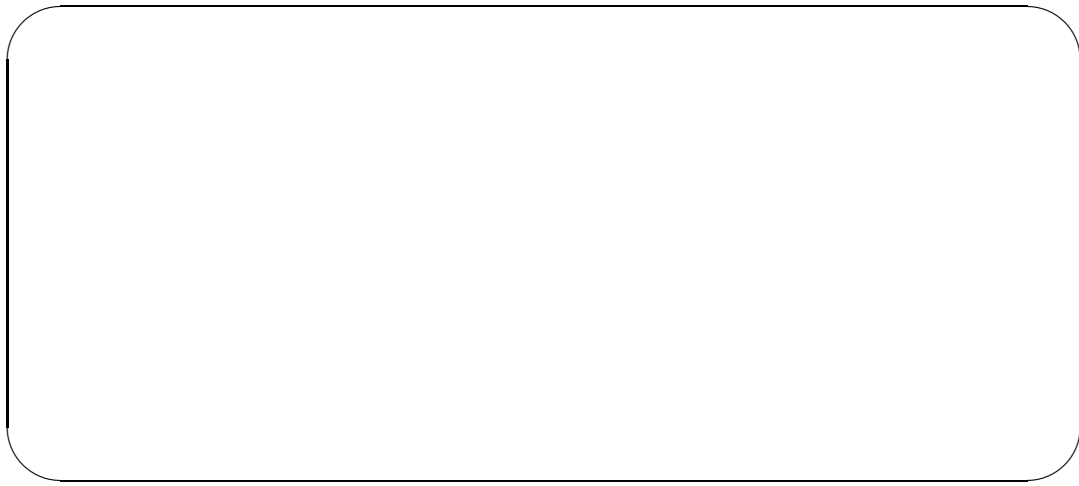
Volem obtenir una funció en C++ que rebi el vector v i retorni l'element dominant de v , o el nombre -1 en cas que no existeixi cap element dominant.

- (a) (2.5 pts.) Un estudiant de PRO2 ens suggereix la següent solució:

```
int dominant_pro2 (vector<int> v) {
    int n = v.size ();
    for (int i = 0; i < n; ++i) {
        if (v[i] != -1) {
            int times = 0;
            int candidate = v[i];
            for (int j = i; j < n; ++j) {
                if (v[j] == candidate) {
                    ++times;
                    v[j] = -1;
                }
            }
        }
    }
}
```

```
        if (times > n/2) return candidate;  
    } } } }  
    return -1;  
}
```

Analitzeu el seu cost en cas pitjor en funció de n . Expliqueu com construiríeu un vector de mida n pel qual es doni aquest cas pitjor.



Analitzeu el seu cost en cas millor en funció de n . Expliqueu com construiríeu un vector de mida n pel qual es doni aquest cas millor.



Si ens asseguren que, per a qualsevol n , el vector v sempre tindrà com a molt 100 naturals diferents, canviaria el seu cost en cas pitjor?

- (b) (2 pts.) Un altre estudiant de *PRO2* se n'adona que si primer ordenem el vector existeix un algorisme ben senzill:

```
int dominant_sort (vector<int> v) {  
    int n = v.size ();  
    own_sort(v.begin (), v.end ());  
    int i = 0;  
    while (i < n) {  
        int times = 0, j = i;  
        while (j < n and v[j] == v[i]){  
            ++times;  
            ++j;  
        }  
        if (times > n/2) return v[i];  
        i = j;  
    }  
    return -1;  
}
```

Si *own_sort* es correspon a una ordenació per inserció, quin és el cost en cas millor i pitjor de *dominant_sort* en funció de n ?

Si *own_sort* implementa un *quicksort*, quin és el cost en cas millor i pitjor de *dominant_sort* en funció de n ?

- (c) (3 pts.) Finalment, un estudiant d'EDA molt aplicat, encara que no brillant, ens suggereix una solució basada en dividir i vèncer. No obstant, s'han perdut parts del codi i us demanem que completeu la següent funció:

```
int times (const vector<int>& v, int l, int r, int x) {
    if (l > r) return 0;
    return (v[l] == x) + times(v, l+1, r, x);
}

int dominant_divide (const vector<int>& v, int l, int r) {
    if (l == r) return v[l];
    int n_elems = (r-l+1);
    int m = (l+r)/2;
    int maj_left = dominant_divide(v, ,  );
    if ( maj_left != -1 and times(  ) > n_elems/2) return  ;
    int maj_right = dominant_divide(v, ,  );
    if ( maj_right != -1 and times(  ) > n_elems/2) return  ;
    return -1;
}

int dominant_divide (const vector<int>& v) {
    return dominant_divide(v, 0, v.size()-1);
}
```

Analitzeu el cost de *dominant_divide* en cas pitjor en funció de n .



Examen Parcial EDA

Duració: 1h 30min

15/04/2021

Problema 1

(5 pts.)

Responen a les següents preguntes:

- (a) (1.25 pts.) Escriviu C o F dins de cada casella indicant si l'afirmació corresponent és certa o falsa, respectivament:

$2^{2n} \in O(2^n)$ ☐

$\log(2n) \in O(\log(n))$ ☐

$2^{2n} \in \Omega(2^n)$ ☐

$\log(2n) \in \Omega(\log(n))$ ☐

$2^{2n} \in \Theta(2^n)$ ☐

$\log(2n) \in \Theta(\log(n))$ ☐

Justifiqueu la vostra resposta:

- (b) (1.25 pts.) Considereu el codi següent:

```
int x = 2;
int y = 1;
while (y ≤ n) {
    y = y + x;
    x = x + 1;
}
```

En funció de n , el seu cost és $\Theta(\text{ })$. Justifiqueu la vostra resposta:

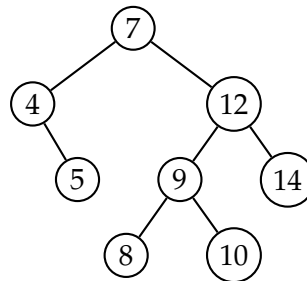
(c) (1.25 pts). Considereu el codi següent:

```
bool f(const map<int,int>& M, const vector<int>& v) {  
    for (int x : v)  
        if (M.find(x) != M.end()) return true;  
    return false;  
}
```

```
int main() {  
    int n; cin >> n;  
  
    map<int,int> M;  
    for (int i = 0; i < n; ++i) {  
        int x; cin >> x;  
        ++M[x];  
    }  
  
    vector<int> v(n);  
    for (int i = 0; i < n; ++i) cin >> v[i];  
  
    cout << f(M,v) << endl;  
}
```

Què fa el codi anterior? Quin és el cost en cas pitjor d'una crida a f en funció d' n ?

- (d) (1.25 pts.) Escriviu l'arbre AVL resultant d'afegir l'element amb clau 11 a l'arbre AVL següent. No cal justificar la resposta.



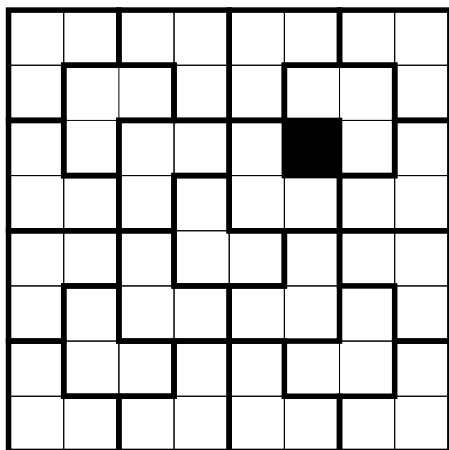
Problema 2

(5 pts.)

Donada una graella de mida $2^n \times 2^n$, amb $n \geq 0$, i que té exactament una casella bloquejada, ens demanen omplir la resta de la caselles amb les següents peces:

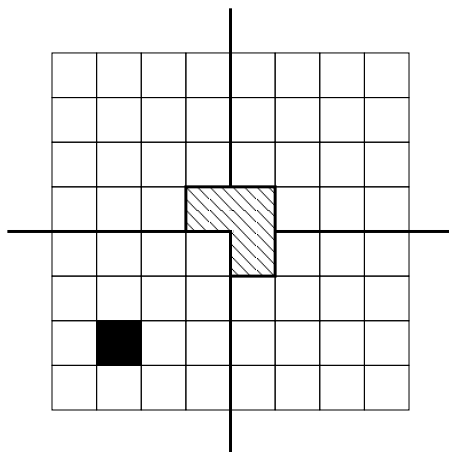


Com és d'esperar, les peces no es poden solapar ni sortir de la graella. Per exemple, per una graella 8×8 i la casella bloquejada en negre, una possible solució és:



- (a) (2 pts.) Demostreu que per a tot $n \geq 0$, sigui quina sigui la posició de la casella bloquejada, sempre podrem omplir la resta de les caselles amb les peces indicades.

Pista: Observeu la següent figura.



- (b) (2 pts.) Completeu el següent codi per tal de resoldre el problema plantejat. La matriu M representa la graella. Les files s'indexen de dalt a baix i les columnes d'esquerra a dreta.

```

void write_sol (const vector<vector<int>>& M);
typedef pair<int,int> Coord;

// Returns quadrant of pos in square [i_l,i_r] x [j_l,j_r]
// Quadrants are:
// 0 1
// 2 3
int quadrant(Coord pos, int i_l, int i_r, int j_l, int j_r) {
    int size = j_r - j_l + 1;
    int i_m = i_l + size / 2;
    int j_m = j_l + size / 2;
    if (  ) return 0;
    if (  ) return 1;
    if (  ) return 2;
    return 3;
}

void fill (vector<vector<int>>& M, int i_l, int i_r, int j_l, int j_r,
          Coord c_blocked, int& num){
    if ( i_l == i_r ) return; // 1x1
    int size = j_r - j_l + 1;
    int i_m = i_l + size / 2; // Midpoints
    int j_m = j_l + size / 2;

    vector<Coord> coords_blocked(4); // Blocked cell in each quadrant
    coords_blocked [0] = {  ,  };
    coords_blocked [1] = {  ,  };
    coords_blocked [2] = {  ,  };
    coords_blocked [3] = {  ,  };
    int q = quadrant(c_blocked, i_l, i_r, j_l, j_r);
    coords_blocked [q] = c_blocked ;

    for (int k = 0; k < 4; ++k)
        if (M[coords_blocked[k].first][coords_blocked[k].second] ==  )
            M[coords_blocked[k].first][coords_blocked[k].second] =  ;
        ++num;

    fill (M,  , num); // Q0
    fill (M,  , num); // Q1
    fill (M,  , num); // Q2
    fill (M,  , num); // Q3
}

int main(){

```

```
int n; cin >> n;
int size = pow(2,n);
Coord blocked;
cin >> blocked.first >> blocked.second;
vector<vector<int>>> M(size,vector<int>(size,-1));

M[blocked.first][blocked.second] = 0;
// 0 initially occupied, -1 to be filled yet, n > 0 indicates piece identifier

int num = 1; // All cells with the same num are part of the same piece
fill (M, 0, size - 1, 0, size - 1, blocked, num);
write_sol (M);
}
```

(c) (1 pt.) Quin és el cost del codi anterior en funció de n ? I en funció del nombre de caselles?

Examen Parcial EDA

Duració: 1h 30min

08/11/2021

Problema 1

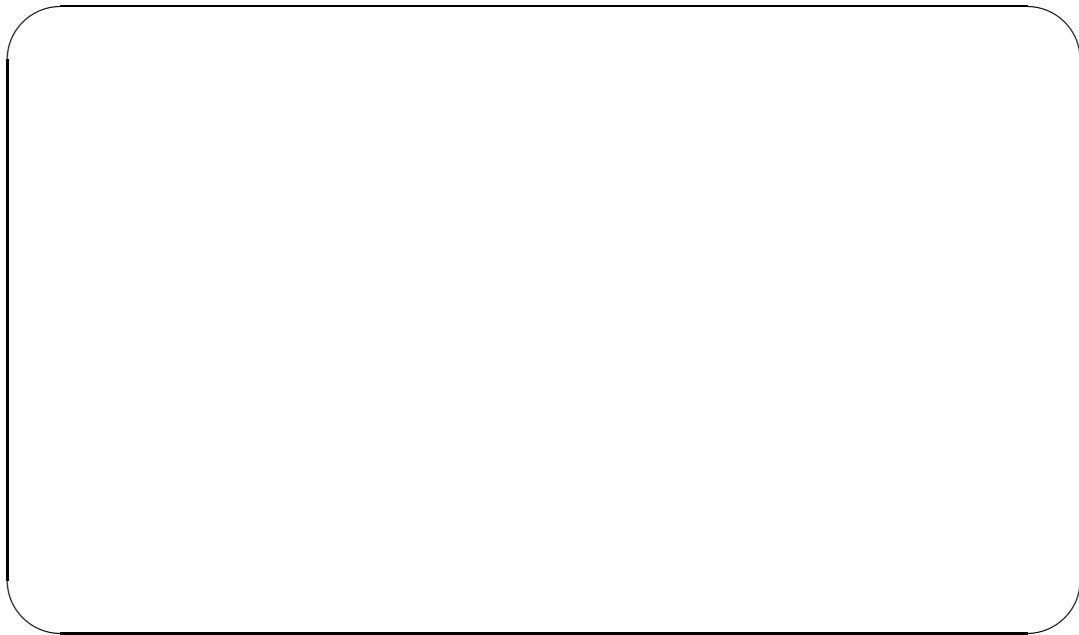
(5 pts.)

Responen les preguntes següents:

(a) (1 pt.) Considereu la funció *mystery*:

```
bool mystery (int n) {  
    if (n ≤ 1) return false;  
    if (n == 2) return true;  
    if (n%2 == 0) return false;  
    for (int i = 3; i*i ≤ n; i += 2)  
        if (n%i == 0) return false;  
    return true;  
}
```

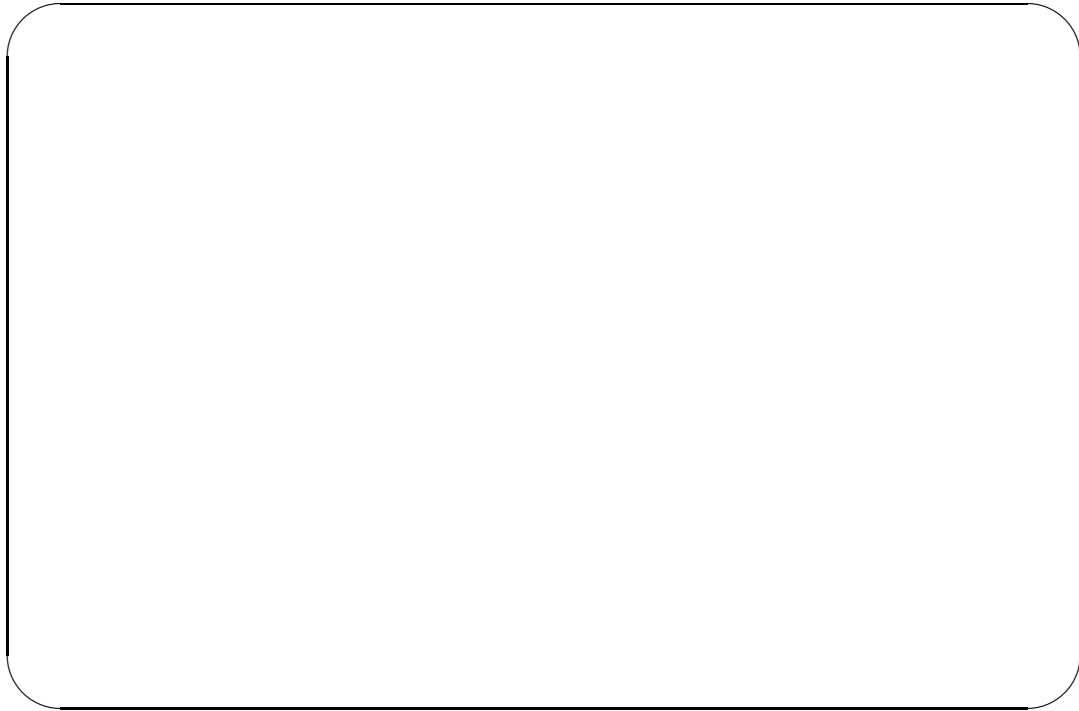
La funció *mystery* determina si i el cost en el cas pitjor, en funció d' n , és $\Theta(\text{ })$. Justifiqueu aquest cost:



(b) (1 pt.) Considereu ara el codi següent:

```
int j = 0;  
int s = 0;  
for (int i = 0; i < n; ++i)  
    if (i == j*j) {  
        for (int k = 0; k < n; ++k) ++s;  
        ++j;  
    }
```

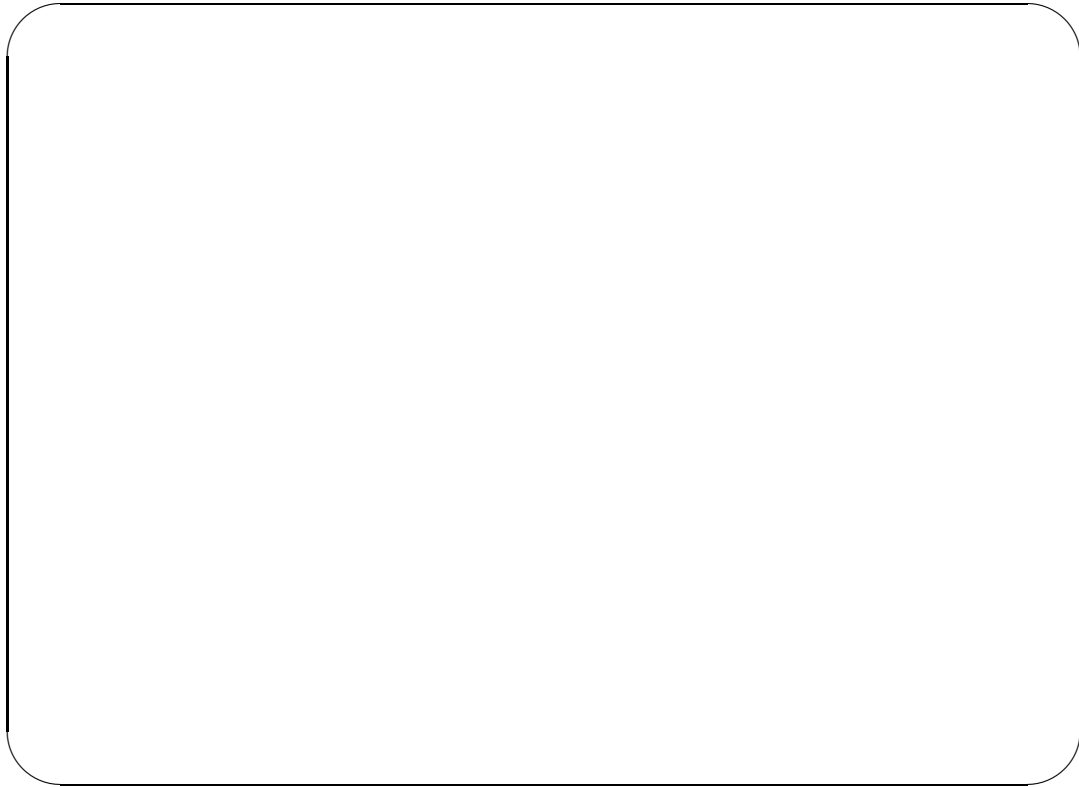
En funció d' n , el cost és $\Theta(\text{ })$. Justifiqueu la vostra resposta:



- (c) (1 pts.) Donat un vector v d' n enters, volem calcular el nombre total de parelles (i, j) tals que $0 \leq i < j \leq n - 1$ i $v[i] = v[j]$. Per tal de solucionar el problema ens donen el codi següent:

```
int pairs (const vector<int>& v, int l, int r) {
    if (l ≥ r) return 0;
    else {
        int m = (l+r)/2;
        int n_left = pairs(v, l, m);
        int n_right = pairs(v, m+1, r);
        int n_crossed = 0;
        for (int i = l; i ≤ m; ++i)
            for (int j = m+1; j ≤ r; ++j)
                if (v[i] == v[j]) ++n_crossed;
        return n_left + n_right + n_crossed;
    }
}
int pairs (const vector<int>& v) {return pairs(v, 0, v.size() - 1);}
```

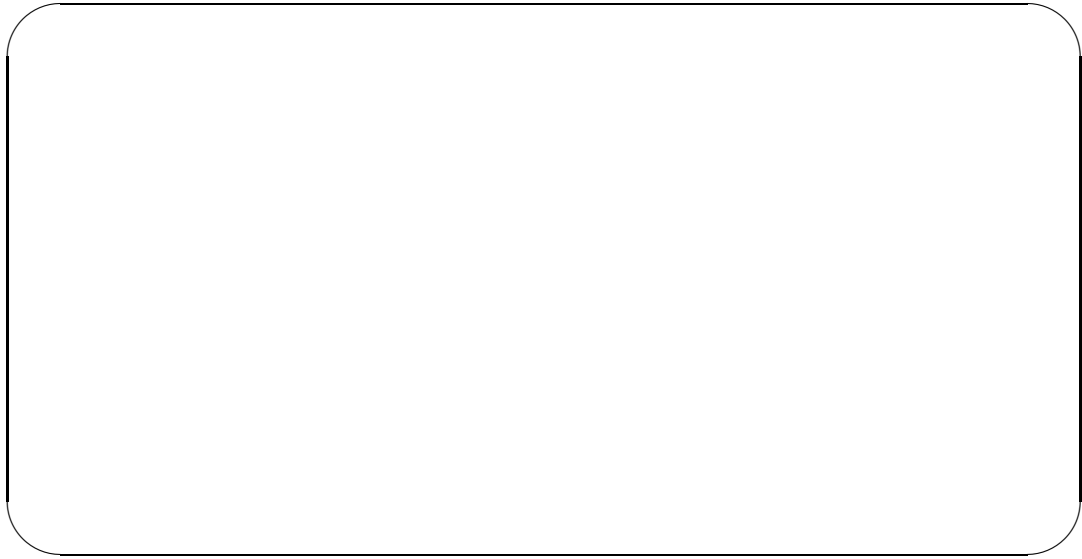
En funció d' n , el cost d'una crida a $\text{pairs}(v)$ és $\Theta(\quad)$. Justifiqueu la vostra resposta:



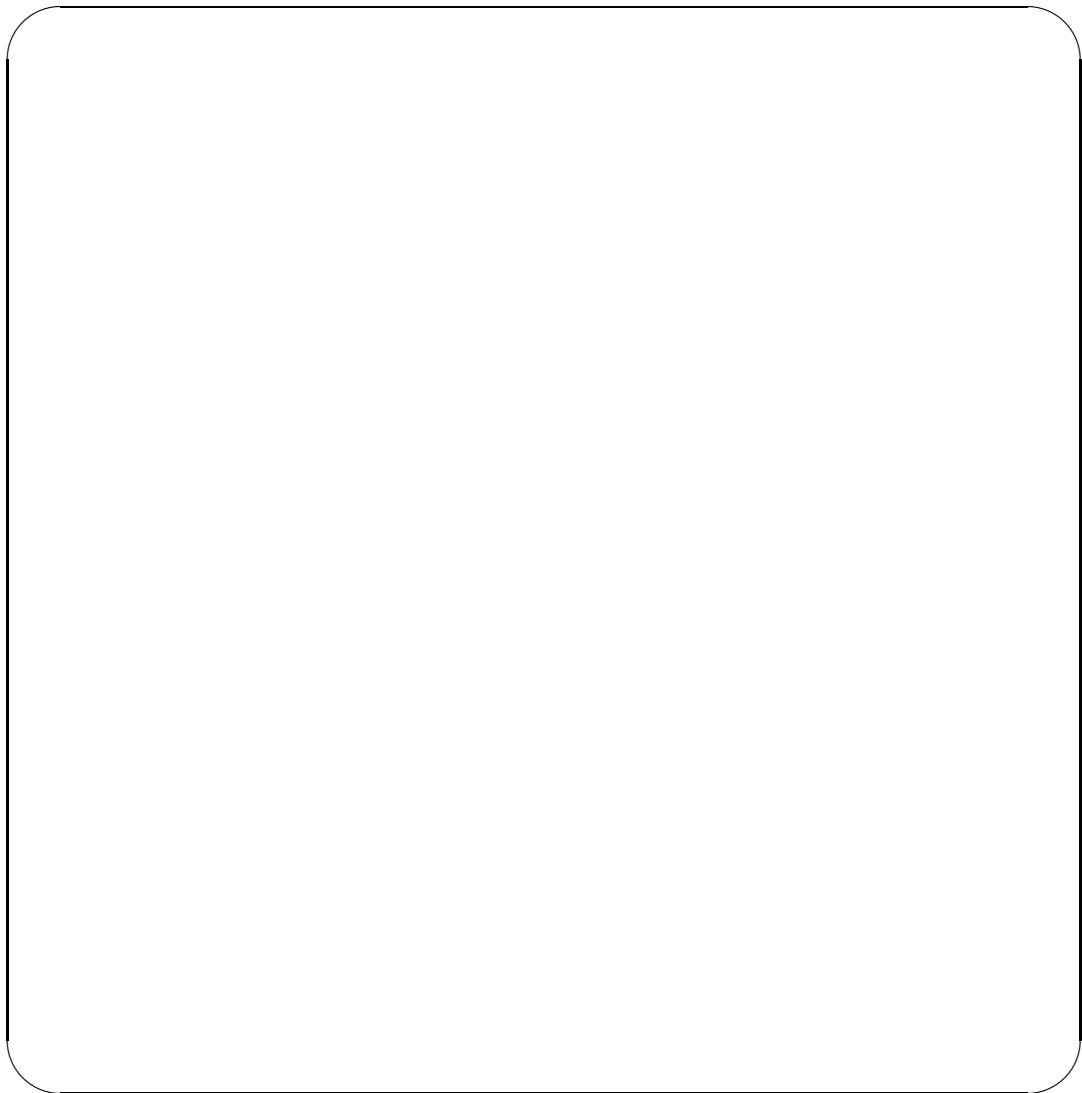
- (d) (2 pts.) Ens asseguren ara que tots els nombres de v són naturals estrictament menors que un cert paràmetre enter K , que és constant i no depèn d' n . En aquesta situació, el codi següent és una solució al problema de l'apartat anterior:

```
int pairs_2 (const vector<int>& v) {  
    vector<int> times(K,0);  
    int n = v.size ();  
    for (int i = 0; i < n; ++i) ++times[v[i]];   
  
    int res = 0;  
    for (int i = 0; i < K; ++i) res += (times[i]*(times[i]-1))/2;  
    return res;  
}
```

Si n és la mida de v , el cost de `pairs_2` en funció d' n és $\Theta(\text{ } \boxed{\text{ }} \text{ })$. Justifiqueu la vostra resposta:

A large, empty rectangular box with rounded corners, intended for the user to paste or write code.

Expliqueu per què el codi anterior és una solució correcta:

A large, empty rectangular box with rounded corners, intended for the user to provide an explanation of the code.

Problema 2**(5 pts.)**

Donat un vector v d' n enters ordenats creixentment i un enter x , volem determinar el nombre de vegades que x apareix a v .

- (a) (1.5 pts.) Diem que la *primera aparició* d' x dins v és el menor índex i amb $0 \leq i < n$ i $v[i] = x$, o bé -1 si x no apareix a v . Un amic ens comenta que si sabem trobar la primera aparició, aleshores trobar una solució eficient al nostre problema no és massa difícil. Ompliu el codi següent per tal que trobi la primera aparició d' x dins v de manera que el seu cost en cas pitjor sigui $O(\log n)$.

```
int first_occurrence (int x, const vector<int>& v, int l, int r) {
    if (l > r) return -1;
    else {
        int m = (l+r)/2;
        if (v[m] < x) return first_occurrence (x,v,m+1,r);
        else if (v[m] > x) return first_occurrence (x,v,l,m-1);
    }
}
```

```
int first_occurrence (int x, const vector<int>& v) {
    return first_occurrence (x,v,0,v.size()-1);
}
```

- (b) (1.5 pts.) Amb la funció anterior funcionant ja correctament, ens demanen que omplim el codi següent per tal que calculi el nombre d'aparicions d' x dins v :

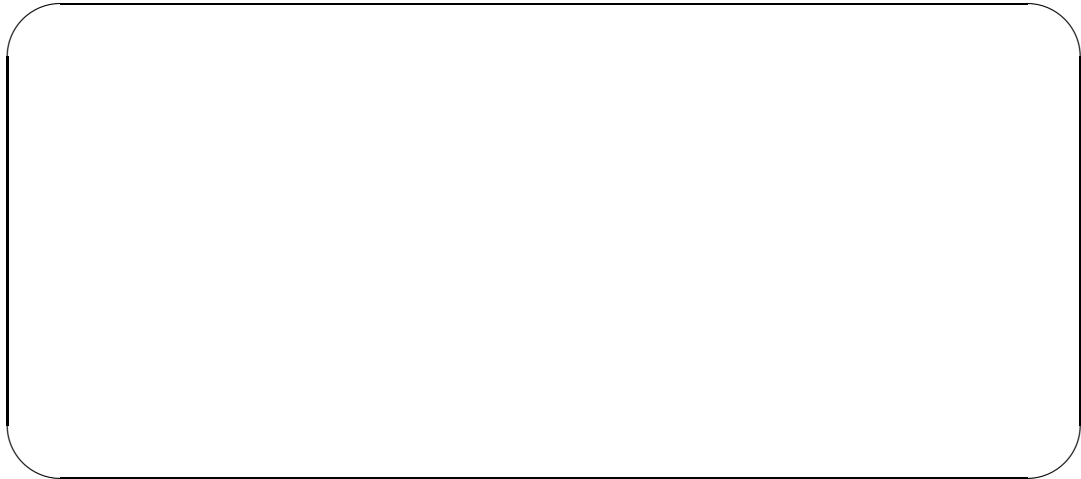
```
int p = first_occurrence (x,v);

int n = v.size ();
for (int i = 0; i < n; ++i) v[i] = -v[i];
for (int i = 0; i <= n/2 - 1; ++i) swap(v[i], v[n-1-i]);
int q = first_occurrence (-x,v);

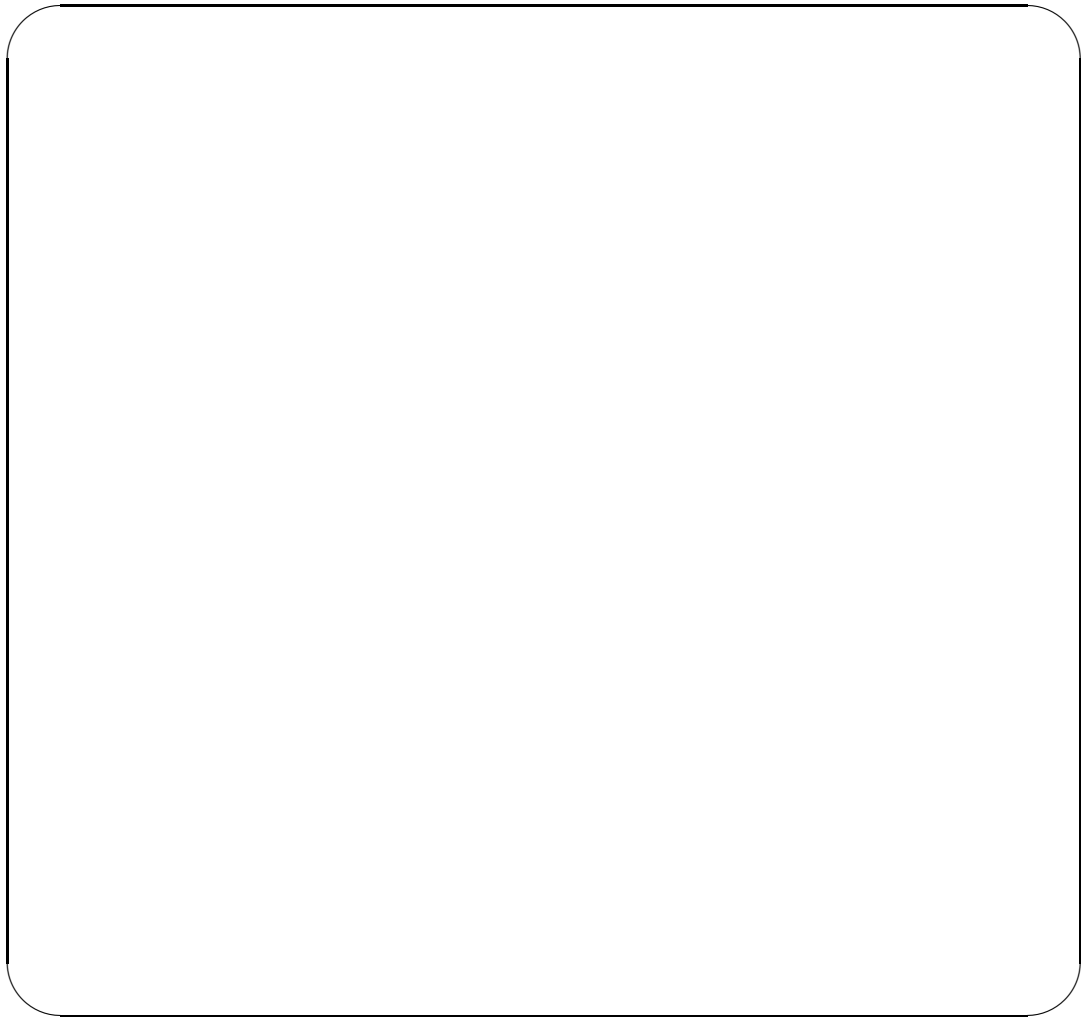
int res;
if (p == -1) res = 0;
else res = 

cout << res << endl;
```

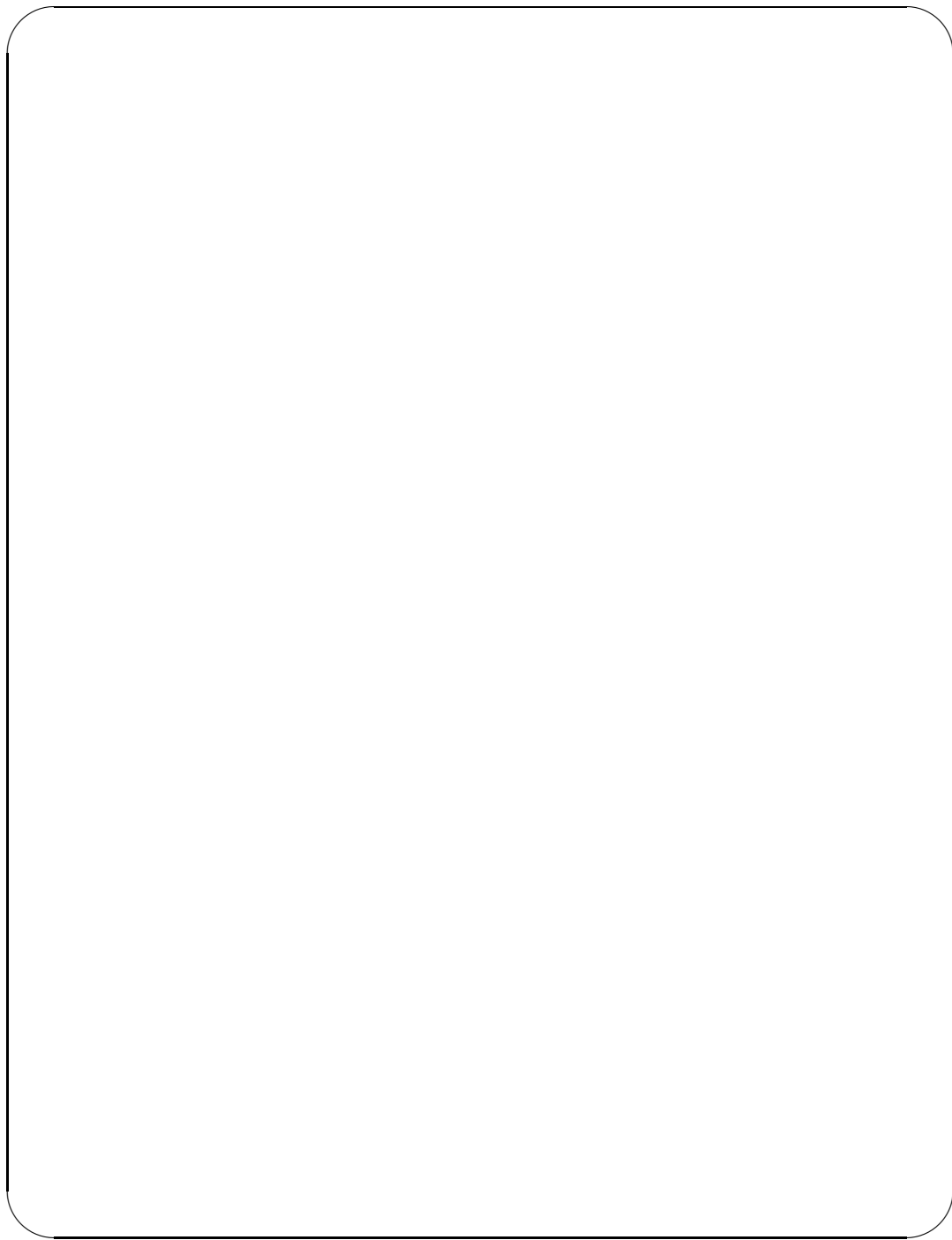
Determineu de forma raonada, el cost en cas pitjor del codi anterior en funció d' n .



Expliqueu per què el codi anterior calcula correctament el nombre d'aparicions d' x dins v .



- (c) (2 pts.) Existeix un algorisme per calcular el nombre d'aparicions que sigui, en cas pitjor, asimptòticament més eficient que el codi anterior? Si existeix, expliqueu-lo a alt nivell (no cal codi concret) i analitzeu el seu cost. Si no existeix, expliqueu per què no pot existir.



Exàmens d'Ordinador

Examen d'Ordinador EDA**Duració: 2 hores****13/12/2010****Torn 1****Problema 1**

Jutge.org, Problema P39846: Tresors en un mapa (4)
(https://www.jutge.org/problems/P39846_ca).

Problema 2

Jutge.org, Problema P71701: Reis pacífics
(https://www.jutge.org/problems/P71701_ca).

Torn 2**Problema 1**

Jutge.org, Problema P84415: La bossa de les paraules
(https://www.jutge.org/problems/P84415_ca).

Problema 2

Jutge.org, Problema P22295: La tortuga viatgera
(https://www.jutge.org/problems/P22295_ca).

Examen d'Ordinador EDA**Duració: 2 hores****19/5/2011****Problema 1**

Jutge.org, Problema P69865: Recollint monedes
(https://www.jutge.org/problems/P69865_ca).

Problema 2

Jutge.org, Problema P98123: Omplint la bossa
(https://www.jutge.org/problems/P98123_ca).

Examen d'Ordinador EDA**Duració: 2 hores****13/12/2011****Problema 1**

Jutge.org, Problema P90766: Tresors en un mapa (3)
(https://www.jutge.org/problems/P90766_ca).

Problema 2

Jutge.org, Problema P67329: ADN
(https://www.jutge.org/problems/P67329_ca).

Examen d'Ordinador EDA**Duració: 2 hores****14/05/2012****Problema 1**

Jutge.org, Problema P47386: Xafardeig
(https://www.jutge.org/problems/P47386_ca).

Problema 2

Jutge.org, Problema P87462: Pacman

(https://www.jutge.org/problems/P87462_ca).

Examen d'Ordinador EDA**Duració: 2 hores****29/11/2012****Problema 1**

Jutge.org, Problema P90861: Cues d'un supermercat (1)

(https://www.jutge.org/problems/P90861_ca).

Problema 2

Jutge.org, Problema P14952: Ordenació topològica

(https://www.jutge.org/problems/P14952_ca).

Examen d'Ordinador EDA**Duració: 2 hores****22/5/2013****Problema 1**

Jutge.org, Problem X07174: Nombres sense prefixos prohibits

(https://www.jutge.org/problems/X07174_ca).

Problema 2

Jutge.org, Problema X34032: El cavall afamat

(https://www.jutge.org/problems/X34032_ca).

Examen d'Ordinador EDA**Duració: 2 hores****4/12/2013****Problema 1**

Jutge.org, Problem P60796: Tresors en un mapa (2)

(https://www.jutge.org/problems/P60796_ca).

Problema 2

Jutge.org, Problema X21319: Valors d'un circuit

(https://www.jutge.org/problems/X21319_ca).

Examen d'Ordinador EDA**Duració: 2 hores****19/5/2014****Problema 1**

Jutge.org, Problema P81453: Camí més curt

(https://www.jutge.org/problems/P81453_ca).

Problema 2

Jutge.org, Problema P89318: Paraules prohibides

(https://www.jutge.org/problems/P89318_ca).

Examen d'Ordinador EDA**Duració: 2 hores****22/12/2014****Problema 1**

Jutge.org, Problema X41530: Bosc

https://www.jutge.org/problems/X41530_ca.**Problema 2**

Jutge.org, Problema X92609: Dues monedes de cada (3)

https://www.jutge.org/problems/X92609_ca.**Examen d'Ordinador EDA****Duració: 2.5 hores****25/05/2015****Problema 1**

Jutge.org, Problema P86108: LOL

https://www.jutge.org/problems/P86108_ca.**Problema 2**

Jutge.org, Problema P27258: Monstres en un mapa

https://www.jutge.org/problems/P27258_ca.**Examen d'Ordinador EDA****Duració: 2.5 hores****22/12/2015****Problema 1**

Jutge.org, Problema P43164: Tresors en un mapa (5)

https://www.jutge.org/problems/P43164_ca.**Problema 2**

Jutge.org, Problema P31389: Torres dins d'un rectangle

https://www.jutge.org/problems/P31389_ca.**Examen d'Ordinador EDA****Duració: 2.5 hores****19/05/2016****Problema 1**

Jutge.org, Problema P12887: Minimum spanning trees

https://www.jutge.org/problems/P12887_en.**Problema 2**

Jutge.org, Problema P54070: Posició d'inserció més a la dreta

https://www.jutge.org/problems/P54070_ca.**Examen d'Ordinador EDA****Duració: 2.5 hores****15/12/2016****Torn 1****Problema 1**

Jutge.org, Problema P76807: Sudoku

https://www.jutge.org/problems/P76807_en.

Problema 2

Jutge.org, Problema P99753: Vector bicreixent
(https://www.jutge.org/problems/P99753_ca).

Torn 2**Problema 1**

Jutge.org, Problema P18679: Barra d'equilibris (1)
(https://www.jutge.org/problems/P18679_ca).

Problema 2

Jutge.org, Problema P74219: Fibonacci numbers (2)
(https://www.jutge.org/problems/P74219_en).

Examen d'Ordinador EDA**Duració: 2.5 hores****25/05/2017****Problema 1**

Jutge.org, Problema P49889: Consonants i vocals (1)
(https://www.jutge.org/problems/P49889_ca).

Problema 2

Jutge.org, Problema P96413: Nombre d'Erdos (2)
(https://www.jutge.org/problems/P96413_ca).

Examen d'Ordinador EDA**Duració: 2.5 hores****11/12/2017****Problema 1**

Jutge.org, Problema P70756: Particions
(https://www.jutge.org/problems/P70756_ca).

Problema 2

Jutge.org, Problema P71496: Retallades
(https://www.jutge.org/problems/P71496_ca).

Examen d'Ordinador EDA**Duració: 2.5 hores****28/05/2018****Problema 1**

Jutge.org, Problema P29033: Dos colors
(https://www.jutge.org/problems/P29033_ca).

Problema 2

Jutge.org, Problema X39049: Potències de permutacions
(https://www.jutge.org/problems/X39049_ca).

Examen d'Ordinador EDA**Duració: 2.5 hores****03/12/2018****Problema 1**

Jutge.org, Problema X82938: Cerca en un vector unimodal
(https://www.jutge.org/problems/X82938_ca).

Problema 2

Jutge.org, Problema X19647: Rutes Barates

(https://www.jutge.org/problems/X19647_ca).

Examen d'Ordinador EDA

Duració: 2.5 hores

20/05/2019

Problema 1

Jutge.org, Problema X64801: És cíclic?

(https://www.jutge.org/problems/X64801_ca).

Problema 2

Jutge.org, Problema P11655: Sumes iguals (3)

(https://www.jutge.org/problems/P11655_ca).

Examen d'Ordinador EDA

Duració: 2.5 hores

02/12/2019

Problema 1

Jutge.org, Problema P65553: Jocs de taula

(https://jutge.org/problems/P65553_ca).

Problema 2

Jutge.org, Problema P98259: Buscant el telecos

(https://jutge.org/problems/P98259_ca).

Examen d'Ordinador EDA

Duració: 2.5 hores

10/06/2020

Problema 1

Un problema escollit aleatòriament d'entre:

Jutge.org, Problema X39187: Comprant accions

(https://jutge.org/problems/X39187_ca).

Jutge.org, Problema X46137: Paraules compensades

(https://jutge.org/problems/X46137_ca).

Jutge.org, Problema X57029: Pujades i baixades

(https://jutge.org/problems/X57029_ca).

Problema 2

Un problema escollit aleatòriament d'entre:

Jutge.org, Problema X14417: Buscador de pel·lícules

(https://jutge.org/problems/X14417_ca).

Jutge.org, Problema X39759: Joc de cavall

(https://jutge.org/problems/X39759_ca).

Examen d'Ordinador EDA**Duració: 2.5 hores****11/01/2021****Torn 1****Problema 1**

Jutge.org, Problema X40596: Seqüències equilibrades
(https://jutge.org/problems/X40596_ca).

Problema 2

Jutge.org, Problema X34137: Vèrtexs intermedis
(https://jutge.org/problems/X34137_ca).

Torn 2**Problema 1**

Jutge.org, Problema X41088: Seqüències sense pous
(https://jutge.org/problems/X41088_ca).

Problema 2

Jutge.org, Problema X50299: Carreteres curtes
(https://jutge.org/problems/X50299_ca).

Examen d'Ordinador EDA**Duració: 2.5 hores****08/06/2021****Problema 1**

Jutge.org, Problema X67572: Paraules concatenades
(https://jutge.org/problems/X67572_ca).

Problema 2

Jutge.org, Problema X13208: Camí en un tauler
(https://jutge.org/problems/X13208_ca).

Examen d'Ordinador EDA**Duració: 2.5 hores****07/01/2022****Torn 1****Problema 1**

Jutge.org, Problema X68591: Component connex màxim i mínim
(https://jutge.org/problems/X68591_ca).

Problema 2

Jutge.org, Problema X18624: Runes
(https://jutge.org/problems/X18624_ca).

Torn 2**Problema 1**

Jutge.org, Problema X83283: Vèrtex més allunyat
(https://jutge.org/problems/X83283_ca).

Problema 2

Jutge.org, Problema X20680: Paraules amb x, y i z
(https://jutge.org/problems/X20680_ca).

3

Exàmens Finals

Examen Final EDA

Duració: 3 hores

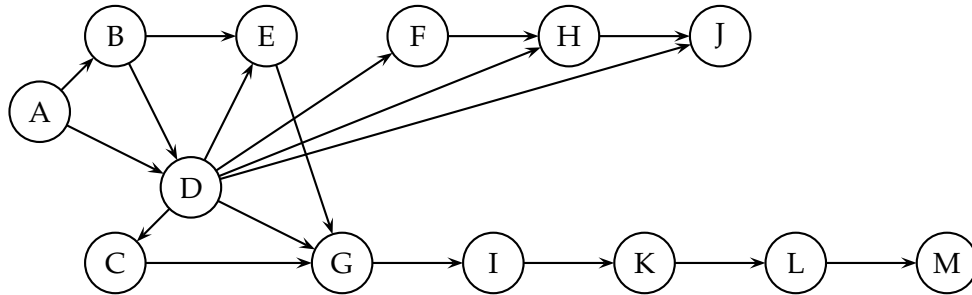
18/01/2011

Problema 1

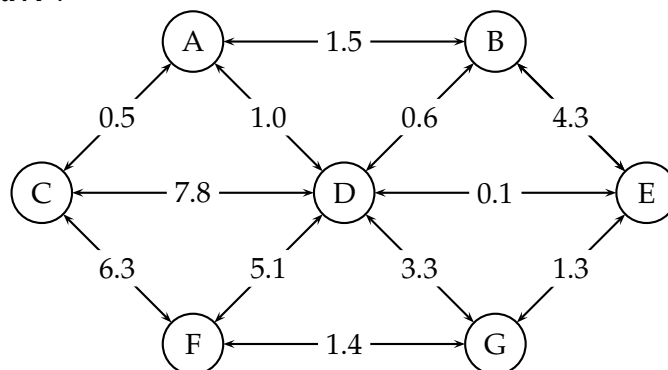
(1,5 punts)

Mostreu els passos que heu seguit per trobar les vostres respostes.

- a) Doneu l'ordenació topològica més petita en ordre alfabètic del graf següent.

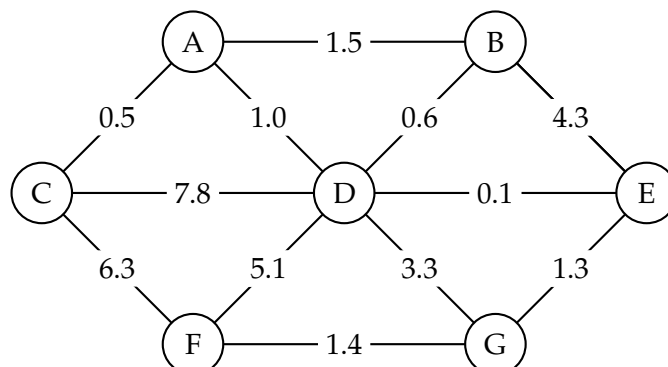


- b) Feu servir l'algorisme de Dijkstra per trobar les distàncies mínimes de A cap a tots els altres vèrtexs del graf següent. Escriviu la seqüència de vectors v , amb $v[X] =$ "distància de A a X".



$v[A]$	$v[B]$	$v[C]$	$v[D]$	$v[E]$	$v[F]$	$v[G]$
0	∞	∞	∞	∞	∞	∞

- c) Feu servir l'algorisme de Prim per trobar un arbre d'expansió mínim del graf següent.



Problema 2**(2 punts)**

Considereu que n és un enter positiu i que G és un graf de N nodes (vèrtexs) i E arestes (arcs) amb pesos. Per cadascuna de les afirmacions següents, digueu amb una petita justificació, si són certes o falses.

- Si f és una funció de la forma $f(n) = An^k$ (amb A i k constants), llavors $f \in O(2^n)$.
- Donats dos vèrtexs u i v de G i un camí mínim C de u a v aleshores, per tot vèrtex w de C , el prefix de C acabat en w és un camí mínim de u a w .
- Si G és connex aleshores l'arbre d'expansió mínim de G conté les $N - 1$ arestes de menys pes.
- Sigui G un graf dirigit i u, v dos dels seus vèrtexs. Si s'afegeix un 1 al pes de cada arc, aleshores el camí mínim de u a v segueix sent mínim.

Problema 3 — Arbres binaris de cerca**(1 punt)**

Considereu la definició de tipus següent per als arbres binaris de cerca:

```
struct Node {  
    int x;           // Informació en el node  
    Node* fe;        // Punter al fill esquerre  
    Node* fd;        // Punter al fill dret  
};  
  
typedef Node* Abc; // Un ABC es denota per un punter a la seva arrel  
                  // (null si és buit)
```

Implementeu en C++ un procediment **void** *escriu* (*Abc a*); que escrigui, en ordre creixent, tots els elements de l'arbre binari de cerca a . Digueu quin és el seu cost.

Problema 4 — Complexitat**(1 punt)**

Considereu els dos problemes següents:

- Donats n objectes amb pesos p_1, \dots, p_n i una alforja amb capacitat suficient, decidir si es poden repartir equitativament aquests objectes en les dues bosses de l'alforja.

Recordeu que una alforja és un sac obert pel mig i tancat pels caps, els quals formen dues bosses grosses, ordinàriament quadrangulars. Aleshores, es demana si es poden repartir tots els objectes entre les dues bosses de manera que cada objecte s'ha de ficar en una bossa i les dues bosses porten exactament la mateixa càrrega.

- Donats n objectes amb pesos p_1, \dots, p_n i una motxilla amb capacitat C , decidir si amb exactament tres objectes es pot carregar la motxilla completament (és a dir, la suma dels pesos dels 3 objectes és C).

Sota la hipòtesi $P \neq NP$ digueu per cadascun d'aquests problemes si pertany o no a la classe P .

Problema 5 — Diccionaris**(2.5 punts)**

Durant la guerra de Vietnam, i abans de ser enviat a Cambodja per assassinar el coronel Kurtz, el capità Willard tenia sota les seves ordres n soldats.

Durant el viatge per la jungla, el capità Willard va constatar que els seus homes podien tenir o no confiança els uns en els altres. Per això, va confeccionar un conjunt $E = \{(u_1, v_1), \dots, (u_m, v_m)\}$ on cada parell (u_i, u_j) indica que el soldat u_i té confiança en el soldat u_j .

El capità Willard va comprendre que la seva missió només podria tenir èxit si la confiança entre els seus soldats era mútua. És a dir, que si u_i confia en u_j , llavors u_j també confia en u_i .

Acostumat a pensar fredament en situacions d'emergència, el capità va entendre que havia de comprovar si el dígraf definit per E era o no simètric. I, evidentment, podria doncs resoldre el seu problema en temps $\Theta(n^2)$. Però les bales ja començaven a caure i necessitava un algorisme més eficient!

Ajudeu en Willard tot proposant un algorisme que, en temps $O(m)$ en el cas mitjà, determini si el dígraf donat per E és o no simètric.

Problema 6 — Programació dinàmica**(2 punts)**

La profunditat $p(T)$ d'un arbre binari T es defineix recursivament així:

$$\begin{cases} p(\emptyset) = 0, \\ p(T) = 1 + \max(p(T_E), p(T_D)) \text{ si } T \neq \emptyset, \end{cases}$$

on \emptyset és l'arbre buit, T_E és l'arbre esquerre i T_D és l'arbre dret de T .

Donat un nombre natural d , es vol calcular quina és la mida (el nombre de nodes) de l'arbre AVL més petit de profunditat exactament d . Representarem aquest valor com $f(d)$.

- a) Demostreu que $f(d) = f(d-1) + f(d-2) + 1$ si $d \geq 2$.
- b) Dissenyeu un algorisme de programació dinàmica eficient que calculi $f(d)$. Quin cost temporal té? I quin cost en espai? Justifiqueu les vostres respostes.

Examen Final EDA

Duració: 3 hores

6/6/2011

Problema 1

(4 punts)

a) Considereu l'algorisme següent:

```

const int K = ...

void desperdici (int n) {
    if (n > 0) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                cout << i << j << n << endl;
            }
        }
        for (int i = 0; i < K; ++i) desperdici (n/2);
    }
}

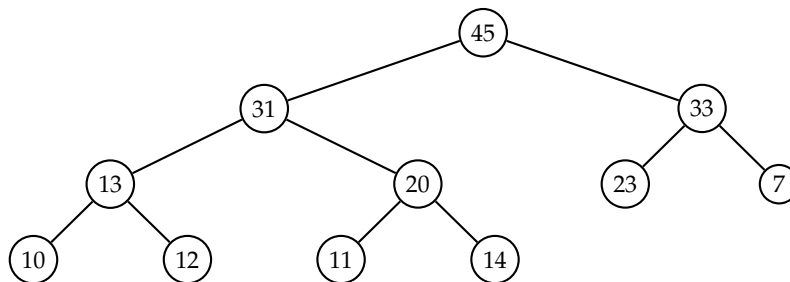
```

Doneu el seu cost en funció de n (segons quin sigui el valor de K). Mostreu els passos que heu seguit.

b) Esteu desenvolupant un algorisme de dividir i vèncer que, per ser útil a la pràctica, ha de tenir un cost asimptòtic $O(n \log n)$. Heu decidit dividir el problema, recursivament, en 3 subproblemes de mida $n/4$, on n és la mida del problema original.

Utilitzant aquesta estratègia és possible aconseguir la complexitat desitjada? Si és el cas, quin és el cost màxim (en funció de n) que es pot utilitzar per dividir el problema original i combinar les solucions dels subproblemes? Expliqueu-ho.

c) El max-heap de la figura següent és el resultat d'una seqüència d'operacions d'inserció i esborrat-del-màxim. La darrera operació va ser una inserció.

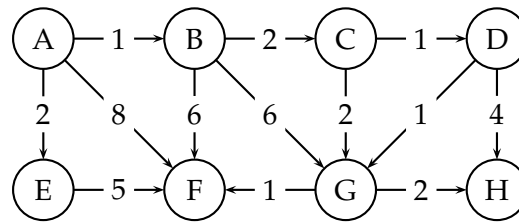


Llisteu les claus que poden ser l'última clau inserida .

Dibuixeu el max-heap resultant d'esborrar el màxim del heap de la figura.

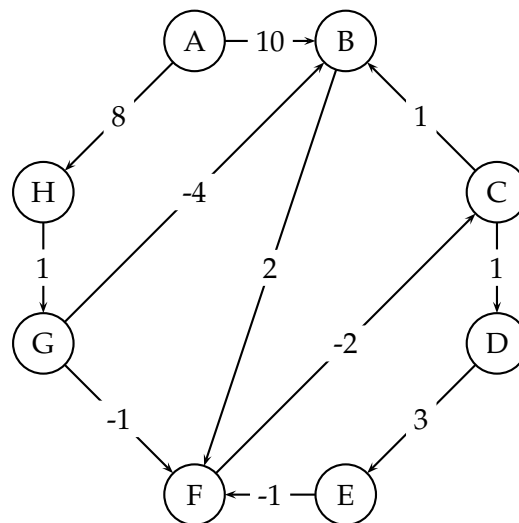
d) Feu servir l'algorisme de Dijkstra per trobar les distàncies mínimes de A cap a tots els altres vèrtexs del graf següent. Escriviu la seqüència de vectors v , amb $v[X] =$

"distància de A a X".



$v[A]$	$v[B]$	$v[C]$	$v[D]$	$v[E]$	$v[F]$	$v[G]$	$v[H]$
0	∞	∞	∞	∞	∞	∞	∞

- e) Feu servir l'algorisme de Bellman-Ford per trobar les distàncies mínimes de A cap a tots els altres vèrtexs del graf següent. Escriviu la seqüència de vectors v , amb $v[X]$ = "distància de A a X".



$v[A]$	$v[B]$	$v[C]$	$v[D]$	$v[E]$	$v[F]$	$v[G]$	$v[H]$
0	∞	∞	∞	∞	∞	∞	∞

Problema 2

(2 punts)

Considereu la definició de tipus següent per als arbres binaris de cerca:

```

struct Node {
    Elem info;      // Informació en el node
    Node* fe;       // Punter al fill esquerre
    Node* fd;       // Punter al fill dret
};

typedef Node* Abc; // Un ABC es denota per un punter a la seva arrel
                  // (null si és buit)

```

Donat un arbre binari de cerca t i un element x , considereu la funció `misteri` següent:

```

int comptar(Abc t){
    if (t == null) return 0;
    return 1 + comptar(t->fe) + comptar(t->fd);
}

int misteri (Abc t, Elem x){
    if (t == null) return 0;
    if (t->info < x) return 1 + comptar(t->fe) + misteri (t->fd, x);
    return misteri (t->fe, x);
}

```

- Digueu què fa la funció `misteri`.
- Digueu justificadament quin és el seu cost, en el cas pitjor, en funció de la mida de l'arbre.
- Considereu, ara, la definició de tipus següent:

```

struct NouNode {
    Elem info;      // Informació en el node
    int m;          // Talla de l'arbre arrelat al node, és a dir,
                  // el nombre de nodes (arrel inclosa)
                  // del subarbre del qual és arrel.
    NouNode* fe;    // Punter al fill esquerre
    NouNode* fd;    // Punter al fill dret
};

typedef NouNode* Abc;

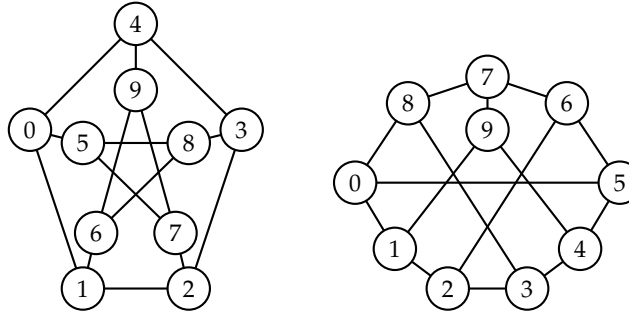
```

Mostreu una nova implementació (més eficient en general) de la funció `misteri` amb aquesta definició de tipus.

- Millora el cost en el cas pitjor de `misteri` amb la nova implementació? Expliqueu-ho.
- Quin és el cost en el cas pitjor de `misteri` amb la nova implementació si l'arbre d'entrada és un AVL? Expliqueu-ho.

Problema 3: Cerca exhaustiva**(2 punts)**

Dos grafs no dirigits $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ es diuen *isomorfs* si existeix una bijecció $f : V_1 \rightarrow V_2$ tal que $\{u, v\} \in E_1$ si i només si $\{f(u), f(v)\} \in E_2$. Per exemple, els dos grafs següents són isomorfs, com ho certifica la bijecció $f(0) = 8, f(1) = 3, f(2) = 2, f(3) = 6, f(4) = 7, f(5) = 0, f(6) = 4, f(7) = 1, f(8) = 5$ i $f(9) = 9$.



Donats dos subconjunts $S_1 \subseteq V_1, S_2 \subseteq V_2$, i una bijecció $f : S_1 \rightarrow S_2$, diem que f és un *isomorfisme parcial* per a S_1 i S_2 quan per a tot parell $u, v \in S_1$ es compleix $\{u, v\} \in E_1$ si i només si $\{f(u), f(v)\} \in E_2$. Observeu que si f és un isomorfisme parcial per a $S_1 = V_1$ i $S_2 = V_2$, llavors G_1 i G_2 són isomorfs.

L'equip de professors d'EDA us ha dissenyat un algorisme de tornada enrera (*backtracking*) per trobar si dos grafs donats són isomorfs o no (i en el cas que ho siguin, retornar una bijecció que ho certifiqui). Malauradament, en enviar-vos el seu programa, part del codi s'ha perdut. Així doncs, completeu la classe següent en C++, tot indicant quina instrucció o condició (només una per espai) posaríeu cada espai requadrat.

Observeu que els grafs (no dirigits) es troben implementats amb matrius d'adjacència. Per tant, $G[u][v]$ és un booleà indicant si u i v es troben connectats en G , i val el mateix que $G[v][u]$.

Observeu també que se suposa que els grafs ja tenen el mateix nombre de vèrtexs i d'arestes (altrament seria immediat concloure que no són isomorfs).

```
typedef matrix<bool> graf;
```

```
class Isomorfisme {
    graf G1, G2;           // els grafs d'entrada
    int n;                 // el nombre de vèrtexs
    vector<int> f;         // la bijecció parcial
    vector<bool> usat;     // usat[w] = (algun v ≤ u compleix f[v] = w)
    bool iso;              // indica si G1 i G2 són isomorfs

    bool backtracking (int u) {
        if (  ) return true;
        for (int w=0; w<n; ++w) {
            if (not usat[w]) {
                f[u] = w;
                usat[w] = true;
            }
        }
    }
};
```



```

        if ( isomorfisme_parcial (u)) if ( backtracking (u+1)) return true;
        [ ]
    }
}
return false;
}

bool isomorfisme_parcial (int u) {
    for (int v=0; v<u; ++v) {
        if ( [ ] ) return false;
    }
    return true;
}

public:
    // Precondició: G1 i G2 tenen el mateix nombre de vèrtexs i d'arestes.
    Isomorfisme (graf G1, graf G2) {
        this->G1 = G1; this->G2 = G2; n = G1.size();
        f = vector<int>(n);
        usat = vector<boolean>(n,false);
        [ ]
    }

    bool isomorfs () { return iso; }

    // Precondició: isomorfs().
    vector<int> bijeccio () { return f; }
}

```

Problema 4 — Complexitat

(2 punts)

Considereu els dos problemes següents:

CLIQUE: Donat un graf $G = (V, E)$ i un natural k , determinar si G conté un subgraf complet de k vèrtexs. Recordeu que un graf complet és aquell que conté totes les arestes possibles entre tots els seus vèrtexs.

CLIQUE-3: És el problema del CLIQUE restringit a grafs en què cada vèrtex té com a molt grau 3.

Sabem que CLIQUE és un problema NP-complet.

- Proveu que CLIQUE-3 pertany a la classe NP.
- Digueu què falla a la prova següent de la NP-completesa de CLIQUE-3.

“Sabem que el problema del CLIQUE per a grafs qualssevol és NP-complet, per tant és suficient donar una reducció polinòmica de CLIQUE-3 a CLIQUE. Donat un graf amb vèrtexs de grau ≤ 3 , i un paràmetre k , la reducció deixa idèntics tant el graf com el paràmetre: clarament el resultat de la reducció és una entrada possible del problema

del CLIQUE. A més, la solució a tots dos problemes és idèntica, cosa que prova la correctesa de la reducció i, per tant, juntament amb la prova de l'apartat a), la NP-completesa de CLIQUE-3."

Examen Final EDA

13/1/2012

Problema 1

(2 punts)

- a) Doneu el cost en el casos pitjor, millor i mitjà de l'algorisme més eficient (en el cas pitjor) que coneguem per construir un max-heap a partir d'una taula donada.

Convertiu en max-heap la taula següent usant l'algorisme de la resposta anterior.

2	9	7	6	5	8	1	8	6	5	3	7	4
---	---	---	---	---	---	---	---	---	---	---	---	---

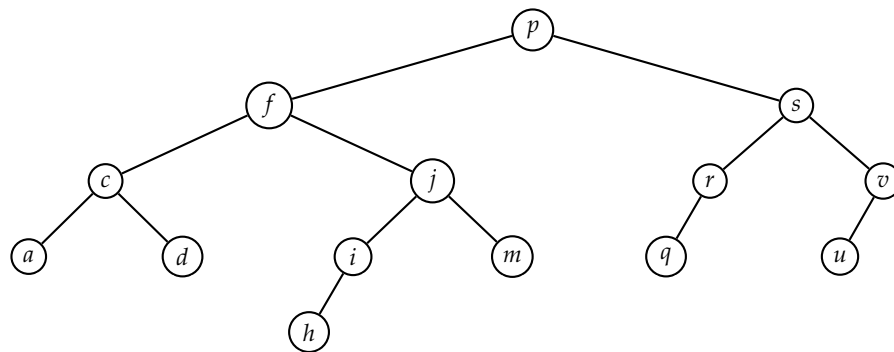
Max-heap:

--	--	--	--	--	--	--	--	--	--	--	--	--

- b) Per a l'alfabet $\{A, B, C, D\}$ i la taula de freqüències següent per a 100 caràcters, doneu un codi de Huffman i dibuixeu l'arbre corresponent.

A	70
B	10
C	15
D	5

- c) Dibueixu l'arbre resultant d'inserir les tres claus x, y, z en aquest ordre a l'arbre AVL de la figura. Noteu que les claus s'ordenen per ordre alfabètic.

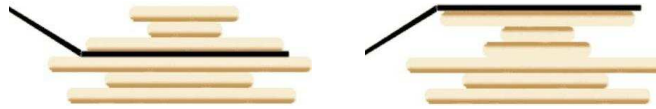


- d) Proposeu, sense donar codi, un algorisme, el més eficient possible, per calcular M^n on M és una matriu de $n \times n$ naturals de n bits cadascun i n és gran.

Problema 2

(2 punts)

L'ordenació per pancakes (una mena de *crêpes* gruixuts) és una variació dels problemes d'ordenació en la qual l'única operació permesa és girar els elements d'un prefix de la taula a ordenar. Aquesta operació es pot visualitzar pensant en una pila de pancakes en la qual es permet agafar els k pancakes superiors amb una espàtula i donar-los la volta d'un sol cop. Vegeu la figura.



(A diferència dels algorismes d'ordenació tradicionals, que intenten ordenar amb el mínim nombre possible de comparacions, l'objectiu de l'ordenació per pancakes és ordenar la seqüència amb el mínim nombre possible de girs.)

Doneu un algorisme per ordenar una taula amb n pancakes utilitzant $2n$ girs o menys.

Pista: inspireu-vos en l'algorisme d'ordenació per selecció.

L'enunciat d'aquest problema es basa en la pàgina:

http://en.wikipedia.org/wiki/Pancake_sorting

Problema 3

(2 punts)

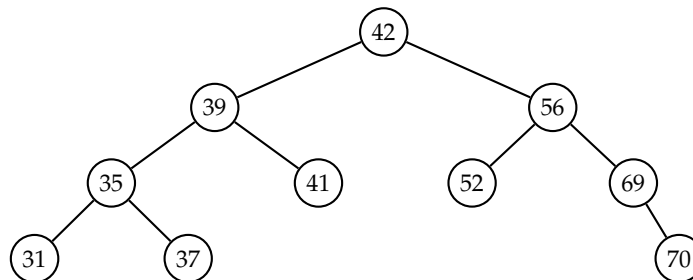
Considereu la definició de tipus següent per als arbres binaris de cerca:

```
struct Node {
    Elem info;           // Informació en el node
    Node* fe;            // Punter al fill esquerre
    Node* fd;            // Punter al fill dret
};

typedef Node* Abc; // Un ABC es denota per un punter a la seva arrel
                  // (null si és buit)
```

Implementeu en C++ i analitzeu en cas pitjor una funció `Node* seguent(Abc a, Elem x)` que, donat un arbre binari de cerca `a` i un element `x`, retorni un punter al node que conté el mínim element d'entre els que són més grans que `x` en `a`. Si `x` és més gran o igual que l'element màxim de l'arbre, retorneu un punter nul.

Per exemple, per a l'arbre de la figura, el següent de 31 hauria de ser un punter a 35, el següent de 32 també hauria de ser un punter a 35, el següent de 69 hauria de ser un punter a 70, i el següent de 70 hauria de ser un punter nul.



Fixeu-vos que x pot no ser a l'arbre i que l'arbre pot ser buit.

Problema 4

(2 punts)

L'equip de professors d'EDA us ha dissenyat un algorisme que, donat un graf dirigit amb costos positius als arcs, i dos vèrtexs x i y , calcula el cost mínim per anar de x a y , i el nombre de maneres d'anar de x a y amb tal cost mínim. Malauradament, en enviar-vos el seu programa, part del codi s'ha perdut. Així doncs, completeu el programa següent en C++, tot indicant quina instrucció o condició (només una per espai) posaríeu en cada espai requadrat.

Observeu que el graf es troba implementat amb llistes d'adjacència. Per tant, $G[u]$ és un vector que conté els successors del vèrtex u , junt amb el cost del corresponent arc.

```
typedef pair<int, int>          ArcP; // arc amb pes
typedef vector< vector<ArcP> > GrafP; // graf amb pesos

const int INFINIT = numeric_limits<int>::max();

void calcula (const GrafP& G, int s, vector<int>& d, vector<int>& p) {
    int n = G.size ();
    d = vector<int>(n, INFINIT);
    
    p = vector<int>(n, 0);
    
    vector<bool> S(n, false);
    priority_queue <ArcP, vector<ArcP>, greater<ArcP> > Q;
    
    while (not Q.empty()) {
        int u = Q.top().second; Q.pop();
        if (not S[u]) {
            S[u] = true;
            for (int i = 0; i < int(G[u].size ()); ++i) {
                int v = G[u][i].first ;
                int c = G[u][i].second;
                if (d[v] > d[u] + c) {
                    d[v] = 
                    p[v] = 
                    
                }
            }
            else if (  ) 
        }
    }
}

int main(void) {
    int n, m;
    while (cin >> n >> m) {
        GrafP G(n);
        for (int k = 0; k < m; ++k) {
```

```

    int u, v, c;
    cin >> u >> v >> c;
    G[u].push_back(ArcP(v,c));
}
int x, y;
cin >> x >> y;
vector<int> d,p;
calcula(G, x, d, p);
if (d[y] == INFINIT)
    cout << "No hi ha cami de " << x << " a " << y << endl;
else
    cout << "Cost " << d[y] << ", " << p[y] << " manera(es)" << endl;
} }

```

Problema 5**(2 punts)**

Considereu els dos problemes següents:

RECOBRIMENT: Donat un graf $G = (V, E)$ i un nombre natural k , determinar si G té un recobriment de les arestes amb k vèrtexs. És a dir, determinar si existeix un subconjunt $A \subseteq V$ de k vèrtexs tal que per cada $\{u, v\} \in E$ tenim que almenys un dels extrems u o v pertany a A .

CLAUS: Donats un nombre natural k , un conjunt de panys \mathcal{P} i un conjunt de claus $\mathcal{C} = \{C_1, \dots, C_n\}$ on cada $C_i \subseteq \mathcal{P}$ representa el subconjunt de panys que la i -èsima clau pot obrir, determinar si existeixen k claus que obrin tots els panys.

Sabem que RECOBRIMENT és un problema NP-complet. Demostreu que CLAUS (altre-ment conegut com *SET COVER*) també ho és.

Pista: Considereu que els vèrtexs són claus i que les arestes són panys.

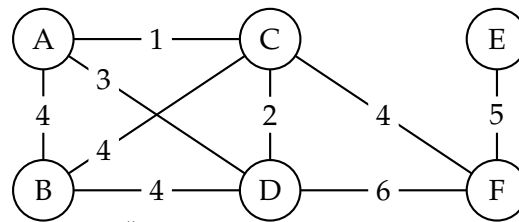
Examen Final EDA

15/6/2012

Problema 1

(3 punts)

- a) (0,5 punts) Proposeu un graf de quatre o més vèrtexs, no dirigit i connex, si existeix, tal que l'ordre en què es visiten els vèrtexs en el recorregut en profunditat sigui idèntic al del recorregut en amplada. Podeu suposar que tots dos recorreguts comencen en el mateix vèrtex, que cal indicar a l'exemple. Si no existeix cap graf indiqueu per què.
- b) (0,5 punts) Dibuixeu un arbre d'expansió mínim del graf de la figura i digueu si és únic.



c) // pre: $0 \leq e \leq d < V.size()$
 void f(int e, int d, vector<int>& V) {
 if (e < d) {
 int m = (e + d)/2;
 intercanvia (V[e], V[m]);
 f(e, m, V);
 intercanvia (V[e], V[m]);
 f(e, m, V);
 intercanvia (V[m+1], V[d]);
 f(m + 1, d, V);
 intercanvia (V[m+1], V[d]);
 }
 }

(0,5 punts) Què fa aquest codi?

(0,5 punts) Quin cost té? Mostreu com heu obtingut la resposta.

- d) Digueu si les afirmacions següents són certes o falses.

- (0,25 punts) En un min-heap de n elements diferents, representat en una taula, l'element màxim ocupa una de les $\lfloor n/2 \rfloor$ posicions finals.
- (0,25 punts) L'algorisme d'ordenació ràpida (quicksort) és l'algorisme (entre els explicats a l'assignatura) d'ordenació més eficient en tots els casos.
- (0,25 punts) En un graf no dirigit el nombre de vèrtexs amb grau senar ha de ser parell.
- (0,25 punts) Un problema decisional és un problema que només té dues possibles respostes: "sí" o "no".

Problema 2 (Taules de dispersió)**(2 punts)**

Considerem la següent implementació d'una taula de dispersió amb encadenament separat semblant a la vista a classe:

```
typedef pair<Key, Info> KeyInfo;
class HashTable {
public:
    int n = 0; // nombre d'elements
    int M = 1; // mida inicial de la taula
    vector< vector<KeyInfo> > v(M); // taula amb encadenament separat

    void insert (const Key& k, const Info& i) {
        int p = hash_function(k)%M;
        v[p].push_back(KeyInfo(k, i));
        double alpha = double(++n)/M;
        if (alpha > 2) resize ();
    }
    ...
}
```

- (1.0 punts) Implementeu l'operació `resize ()` de l'última línia: assumint que la taula de dispersió té factor de càrrega $\alpha > 2$, una crida a `resize ()` la redimensiona per a què tingui factor de càrrega $\alpha = 1$. Recordeu que $\alpha := n / M$.
- (0.5 punts) Assumint que tant `hash_function ()` com `push_back()` tenen cost $\Theta(1)$ i que el factor de càrrega α satisfà la pre-condició $\alpha > 2$, quin cost té, en el cas pitjor, la vostra implementació de `resize ()` en funció del nombre d'elements n ? Justifiqueu la resposta.
- (0.5 punts) Assumint que tant `hash_function` com `push_back()` tenen cost $\Theta(1)$, quin cost té, en el cas pitjor, inserir consecutivament n elements en una taula inicialment buida fent servir l'operació `insert` i la vostra implementació de `resize`? Justifiqueu la resposta.

Problema 3**(2 punts)**

Un arbre binari de cerca per a enters ve definit pel tipus següent:

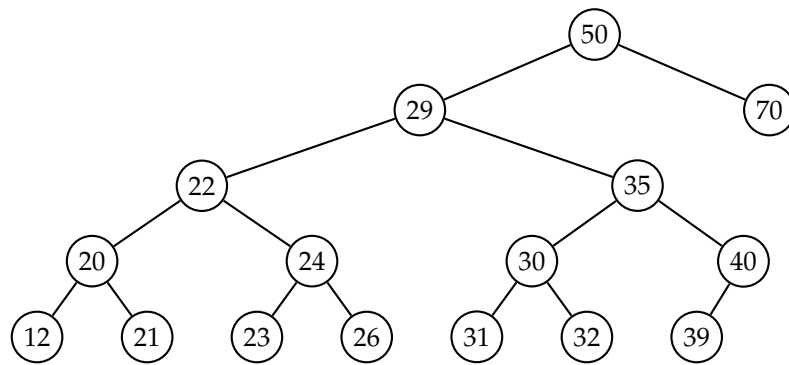
```
struct Node {
    Node* fe;
    Node* fd;
    int x;
};
```

- (1.0 punts) Implementeu la funció

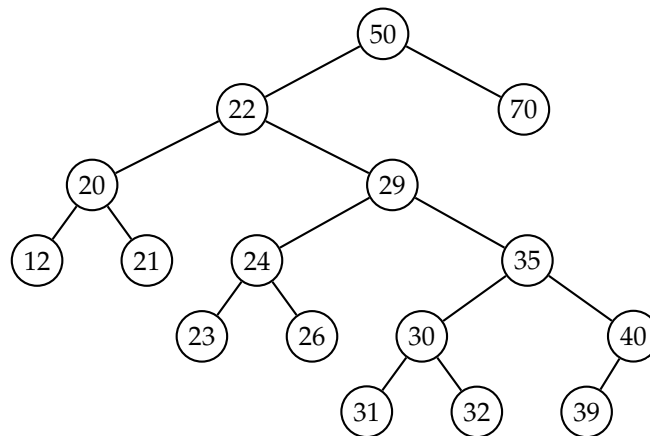
```
void rotacio (Node*& a, int x);
```

que, donats l'arrel d'un arbre binari de cerca a i un element x , apliqui una rotació com la de l'exemple (d'abaix) al node de a que conté x . Podeu suposar que a sempre conté un node amb x , i que aquest node té prou descendents com per poder-hi aplicar la rotació.

Per exemple, el resultat d'aplicar la rotació a l'element 29 d'aquest arbre



dóna aquest arbre



- b) (0.5 punts) Digueu quin és el cost de la funció `rotacio` en el cas pitjor (explicitau quin és el cas pitjor) en funció del nombre de nodes n de l'arbre.
- c) (0.5 punts) Contesteu de nou la pregunta b) suposant que a és un arbre AVL.

Problema 4

(2 punts)

L'equip de professors d'EDA ha dissenyat un programa de backtracking que resol el problema següent: donat un conjunt de n enters positius, calcula la diferència mínima que es pot obtenir "partint" el conjunt original en dos subconjunts i comparant les sumes dels seus elements. Per exemple, donat el conjunt d'entrada $\{3,4,2\}$, la diferència mínima és 1 i es pot obtenir amb la partició: $\{3,2\}$ i $\{4\}$. En canvi, amb el conjunt d'entrada $\{3,4,1,1,1\}$ la diferència mínima és 0 i es pot obtenir amb la partició: $\{3,1,1\}$ i $\{4,1\}$. Malauradament, en enviar-vos el programa, part del codi s'ha perdut. Així doncs, completeu el programa següent en C++, tot indicant quina instrucció o condició (només una) posaríeu en cada espai requadrat.

```
class Particio {
    int n, min_dif, dif, total;
    vector<int> v;
```

```

void recursiu (int i) {
    int va = valor_absolut (dif);
    if (  ) return;
    if ( i == n )  ;
    else {
        dif += v[i]; total -= v[i];
        recursiu (i+1);
         ;
        recursiu (i+1);
        dif += v[i]; total += v[i];
    }
}

```

```

public:
    Particio (int n, int total , vector<int>& v) {
        this -> n = n;
        this -> total = total ;
        this -> v = v;
        min_dif = total ;
        dif = 0;
        recursiu (0);
        cout << min_dif << endl;
    }
};

```

```

int main() {
    int n;
    cin >> n;
    int total = 0;
    vector<int> v(n);
    for (int i = 0; i<n; ++i) { cin >> v[i];  ;}
    Particio p(n, total ,v);
}

```

Problema 5

(1 punt)

Sigui A un problema del qual se sap que és NP-complet, i B un problema del qual només se sap que pertany a NP. Sigui X un problema qualsevol. Responen raonadament si les afirmacions següents són certes o no:

- (0.25 punts) Supposeu que X es pot reduir polinòmicament a A . Pertany X a NP? És X NP-complet?
- (0.25 punts) Supposeu que A es pot reduir polinòmicament a X . Pertany X a NP? És X NP-complet?
- (0.25 punts) Supposeu que X es pot reduir polinòmicament a B . Pertany X a NP? És X NP-complet?

- d) (0.25 punts) Supposeu que B es pot reduir polinòmicament a X . Pertany X a NP? És X NP-complet?

Examen Final EDA**10/01/2013****Problema 1****(2 punts)**

Donada una seqüència de $n > 0$ enters, on cada element apareix dos cops, excepte un element que apareix un sol cop, proposeu un algorisme¹ tan eficient com sigui possible per trobar l'element no repetit. Analitzeu el cost del vostre algorisme en temps i en espai.

Per exemple, per a $\{4, 99999, 1, 1, 4, 2, -5, -1000, -1000, 99999, -5\}$, caldria retornar 2.

Problema 2**(2 punts)**

Donat un vector $V[1..n]$ d'enters, volem trobar els m elements més petits de V .

- Considereu les següents opcions. Digueu en quins casos és millor una, i en quins casos és millor l'altra, segons el seu cost en temps en el cas pitjor.
 - a) Ordenar V i triar els m primers elements del vector ordenat.
 - b) Seleccionar el primer element de V , després el segon, i així successivament (recordeu que seleccionar l'element k -èsim té cost lineal en el nombre d'elements de la selecció).
- Proposeu un mètode² millor que a) i b).

Problema 3**(2 punts)**

Considereu les següents implementacions del tipus "digraf" (graf dirigit):

```
class Digraf1 {
    vector< vector<bool> > > t;

public:

    // Crea un digraf amb vèrtexos 0 ... n - 1 i sense arcs
    Digraf1(int n) { t = vector< vector<bool> > >(n, vector<bool>(n, false)); }

    // Afegeix l'arc u → v
    void afegeix_arc (int u, int v) { t[u][v] = true; } };

class Digraf2 {
    vector< vector<int> > > t;

public:

    // Crea un digraf amb vèrtexos 0 ... n - 1 i sense arcs
    Digraf2(int n) { t = vector< vector<int> > >(n); }

    // Afegeix l'arc u → v
    void afegeix_arc (int u, int v) { t[u].push_back(v); } };
```

¹Descripció d'alt nivell sense implementació.

²Descripció d'alt nivell sense implementació.

En aquest exercici, donat un dígraf $G = (V, E)$ denotarem $n = |V|$ i $m = |E|$. Quan es demani un cost, cal donar la resposta més precisa possible, usant la notació $\Theta(\cdot)$. Quan es demani implementar, cal fer-ho en C++.

1. Quin és el cost en espai en termes de n i m en el cas pitjor d'un dígraf implementat:

- (a) amb la classe *Dígraf1*.
- (b) amb la classe *Dígraf2*.

Quan es dóna aquest cas pitjor, respectivament?

2. Volem afegir un nou mètode **bool** *hi_ha_arc*(**int** u , **int** v) que retorni, donats un dígraf i dos vèrtexos u i v , si hi ha l'arc $u \rightarrow v$. Implementeu aquest mètode i doneu-ne el seu cost temporal en el cas pitjor en termes de n i m si el dígraf està implementat:

- (a) amb la classe *Dígraf1*.
- (b) amb la classe *Dígraf2*.

Quan es dóna aquest cas pitjor, respectivament?

3. Implementeu una classe *Dígraf3* amb les mateixes funcionalitats que les anteriors, i amb un mètode **bool** *hi_ha_arc*(**int** u , **int** v) que tingui cost temporal $O(\log n)$ en el cas pitjor. El cost en espai de representar un graf ha de ser $O(n + m)$.

Problema 4

(2 punts)

Completeu el programa següent de tal manera que decideixi si un graf G donat és 3-colorable.

```
#include <vector>
#include <iostream>

using namespace std;

typedef vector<vector<int>>> Graph;

class ThreeColoring {
    Graph G;
    int n;
    vector<int> col;
    bool done;

    bool legal (int u, int c) {
        
    }

    void rec (int u) {
        if (u == n) {
```

```

        [ ] ;
    } else {
        for (int c = 0; c < 3 and not done; ++c) {
            if (legal(u, c)) {
                col[u] = c;
                [ ] ;
            }
        }
    }
}

```

public:

```

ThreeColoring (Graph G)
:   G(G),
    n(G.size()),
    col(n),
    done(false)
{
    [ ] ;
}

bool colorable () const {
    return done;
}

int color (int u) {
    return col[u];
}

};

int main() {
    Graph G = {
        {1,2},
        {0,3},
        {0,3},
        {1,2}
    };

    ThreeColoring col(G);
    cout << col.colorable () << endl;
}

```

Problema 5

(2 punts)

Considereu els problemes següents:

CLICA: Donat un graf $G = (V, E)$ i un nombre natural k , determinar si G té una k -clica, és a dir, si existeix un subconjunt $U \subseteq V$ de cardinalitat k i per a tot parell de vèrtexs $i, j \in U$ diferents, $\{i, j\} \in E$.

CLICA-CENTRADA: Donat un graf G , un natural k i un vèrtex u de G , determinar si G té una k -clica que conté u .

Es vol demostrar $\text{CLICA} \leq^p \text{CLICA-CENTRADA}$ i, per fer-ho, es consideren consecutivament tres reduccions. Digueu, per a cadascuna d'elles, si és correcta o no, proporcionant un argument si ho és o un contraexemple en cas contrari. Per a tot graf G i tot natural k , es defineix:

- (a) $f(G, k) = (G, k, v)$, on v és un vèrtex de grau màxim de G .
- (b) $g(G, k) = (G', k, w_1)$, on $G' = (V', E')$ consisteix en una còpia de G més una k -clica, és a dir, que per a k vèrtexs nous $w_1, \dots, w_k \notin V$, es defineix

$$V' = V \cup \{w_1, \dots, w_k\}$$

$$E' = E \cup \{\{w_i, w_j\} \mid 1 \leq i < j \leq k\}.$$

- (c) $h(G, k) = (G'', k+1, u)$, on u és un vèrtex nou ($u \notin V$) i $G'' = (V'', E'')$ consisteix en una còpia de G més aquest vèrtex nou connectat a tots els de la còpia de G , és a dir

$$V'' = V \cup \{u\}$$

$$E'' = E \cup \{\{u, v\} \mid v \in V\}.$$

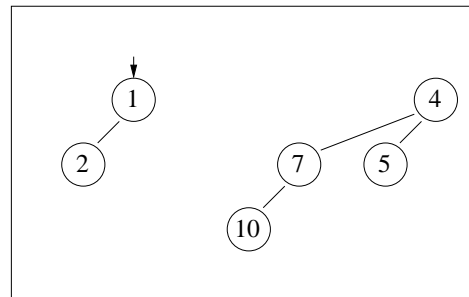
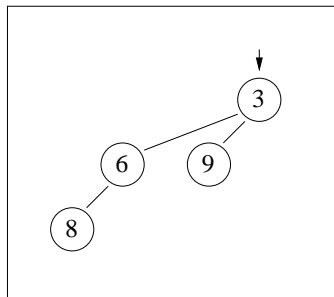
Examen Final EDA**Duració: 3 hores****5/6/2013****Problema 1: Algorisme de Gauss****(2 punts)**

L'algorisme clàssic de Gauss per solucionar sistemes de n equacions lineals amb n incògnites requereix, essencialment, unes $(2n^3 + 3n^2 - 5n)/6$ multiplicacions, unes $n(n-1)/2$ divisions, i unes $(2n^3 + 3n^2 - 5n)/6$ restes. Cap operació elemental s'executa més vegades que les tres operacions aritmètiques considerades.

- Quin és el cost asimptòtic en funció de n ? (0,5 punts)
- Si N denota la talla de l'entrada, quin és el cost en funció de N ? (0,5 punts)
- Si l'algorisme de Gauss triga 15 segons per solucionar un sistema amb 500 equacions i 500 incògnites en un cert ordinador, quant esperaries que trigaria, aproximadament, per solucionar un sistema de 1000 equacions i 1000 incògnites? (0,5 punts)
- Acaben de regalar-te un ordinador 1000 cops més ràpid que el del punt c). Per quin factor s'incrementarà el nombre d'equacions i d'incògnites n que l'ordinador nou serà capaç de resoldre en el mateix temps que l'ordinador vell? (0,5 punts)

Problema 2: Heaps binomials**(1 punt)**

Dibuixeu el heap binomial que resulta d'aplicar l'operació de fusió (també anomenada unió) entre els dos heaps binomials següents:



Resposta (mínimament raonada):

(1 punt)

Problema 3: Potències de grafs**(2 punts)**

Donat un graf dirigit $G = (V, E)$, es defineix la potència k -èsima de G com el graf $G^k = (V, E^k)$, on $(u, v) \in E^k$ si i només si hi ha algun camí de u a v de longitud menor o igual que k (és a dir, amb com a molt k arcs) en el graf G .

- Sigui A la matriu d'adjacències de G considerant que $A[i, i] = 1$ per a tot vèrtex i (i naturalment $A[i, j] = 1$ si i només si $(i, j) \in E$, per $i \neq j$). Es pot veure fàcilment que si en el producte de matrius canviem el producte d'enters per un AND de booleans i la suma per un OR de booleans, aleshores A^2 és la matriu d'adjacències de G^2 .

Demostreu que A^k és la matriu d'adjacències de G^k per a tot $k \geq 1$. (0,5 punts)

- Dissenyeu un algorisme que, donat un graf dirigit $G = (V, E)$ representat per una matriu d'adjacències, calculi $G^4 = (V, E^4)$ en temps $\Theta(n^3)$, on n és el nombre de vèrtexs. (1 punt)

- c) Es pot calcular en temps inferior a $\Theta(n^3)$? Expliqueu com. (0,5 punts)

Problema 4: Reduccions

(2 punts)

Considereu els següents problemes (on el primer és el problema de la clíca de la col·lecció de problemes):

- (a) CLICA: Donats un graf no dirigit $G = (V, E)$ i un natural k , decidir si G conté un subgraf complet de mida k . Formalment, decidir si existeix $U \subseteq V$ tal que $|U| = k$ i per a tot parell $u, v \in U$ amb $u \neq v$ tenim que $(u, v) \in E$.
- (b) CLICA⁺¹: Donats un graf no dirigit $G = (V, E)$ i un natural k , decidir si G conté un subgraf complet de mida $k + 1$. Formalment, decidir si existeix $U \subseteq V$ tal que $|U| = k + 1$ i per a tot parell $u, v \in U$ amb $u \neq v$ tenim que $(u, v) \in E$.

Considereu ara els algorismes A i B següents:

- (i) A : Amb entrada el parell $\langle G, k \rangle$, on $G = (V, E)$ és un graf i k és un natural, retorna el parell $\langle G, k + 1 \rangle$.
- (ii) B : Amb entrada el parell $\langle G, k \rangle$, on $G = (V, E)$ és un graf i k és un natural, retorna el parell $\langle G', k \rangle$ on $G' = (V', E')$ és un graf que obtenim a partir de G afegint un nou vèrtex v_{nou} , $V' = V \cup \{v_{nou}\}$ i afegint una aresta (v_{nou}, v) per a cada $v \in V$, $E' = E \cup \{(v_{nou}, v) | v \in V\}$.

Un d'aquests algorismes redueix CLICA a CLICA⁺¹ i l'altre redueix CLICA⁺¹ a CLICA. Sabeu dir quin és quin?

L'algorisme redueix CLICA a CLICA⁺¹. Justifiqueu-ho. (1 punt)

Problema 5: Nano–Google

(3 punts)

Es vol dissenyar un sistema que permeti recuperar eficientment documents de text a través de consultes sobre les paraules que contenen.

Com a exemple, suposeu que els documents (identificats per enters arbitraris i escrits sense accents) són

- 11: *cada dia al mati canta el gall quiriquire*
 23: *el nen canta a la gallina*
 33: *la gallina canta i el gall dorm*
 41: *la merda de la muntanya no fa pudor*

Llavors, el resultat de la consulta $[el, gall, canta]$ hauria de ser la llista $[11, 33]$ perquè aquests documents són els que contenen totes les paraules de la cerca. Igualment, el resultat de la consulta $[canta]$ hauria de ser la llista $[11, 23, 33]$ i el resultat de la consulta $[guinardo]$ hauria de ser la llista buida $[\]$. Els resultats de les cerques s'han de donar en ordre creixent de número de document. Els documents estan escrits en LAPAO.

El sistema funciona en dues fases. En la primera fase (offline), es pre-processen d'alguna forma els documents. A la segona (online), es reben les consultes i es lliuren les respostes. S'espera que hi hagi molts documents, tots amb bastantes paraules. Es vol contestar molt ràpid cada consulta. Cadascuna de les consultes sol tenir molt poques paraules.

Fixem la nomenclatura següent:

- M = nombre de documents
- N = nombre total de paraules en tots els documents
- n = nombre de paraules diferents en tots els documents
- D = el nombre màxim de documents on apareix cada paraula
- k = el nombre màxim de paraules en una consulta

Nota: Llegiu totes les preguntes abans de començar. Les solucions d'aquest problema amb cost $\Omega(N)$ per a d) són insuficients i no es valoraran.

Part 1: Considereu el cas on cada consulta conté una sola paraula ($k = 1$).

- a) Expliqueu quines estructures de dades i quin pre-procés utilitzaríeu per a la primera fase. Expliqueu com utilitzaríeu aquesta estructura de dades per processar cada consulta. (1 punt)
- b) Quantifiqueu l'espai necessari per l'estructura de dades de la vostra solució. (0,25 punts)
- c) Quantifiqueu el temps necessari per la primera fase (pre-procés) amb la vostra solució. (0,5 punts)
- d) Quantifiqueu el temps necessari per processar una consulta amb la vostra solució. (0,25 punts)

Part 2: Considereu ara el cas general ($k \geq 1$).

- e) Expliqueu com utilitzaríeu la mateixa estructura de dades de la Part 1 per processar cada consulta. (0,5 punts)
- f) Quantifiqueu el temps necessari per processar una consulta amb aquesta solució. (0,5 punts)

Examen Final EDA

Duració: 3 hores

15/1/2014

Problema 1: Preguntes curtes

(2 punts)

- Sigui n parell. El cost de l'algorisme d'ordenació per inserció amb l'entrada

$$[2, 1, 4, 3, 6, 5, \dots, 2i, 2i - 1, \dots, n, n - 1]$$

és $T(n) = \Theta(\quad)$.

- Indiqueu **totes** les afirmacions que siguin correctes: El cost de mergesort en el cas pitjor és: ☐ $O(n^2)$; ☐ $\Theta(n \log n)$; ☐ $\Omega(n \log n)$.
- Doneu una funció $f(n)$ que sigui alhora $\Omega(n)$ i $O(n \log n)$, però que no sigui Θ de cap de les dues. Resposta: $f(n) = \quad$
- Resoleu $T(n) = 3T(n/3) + \Theta(n^2)$. Resposta: $T(n) = \Theta(\quad)$.
- La **recurrència** que expressa el cost de l'algorisme de Karatsuba per multiplicar dos nombres de n dígitos és $T(n) = \quad$.
- Sigui x un vector de n bits qualssevol. Interpretant x com un nombre natural escrit en binari, el cost en cas pitjor de l'algorisme obvi per sumar 1 a x és $T(n) = \Theta(\quad)$.
- Sigui x un vector de n bits escollits uniformement i independentment a l'atzar. Interpretant x com un nombre natural escrit en binari, el cost esperat de l'algorisme obvi per sumar 1 a x és $T(n) = \Theta(\quad)$.
- Considereu l'algorisme següent de cost $\Theta(\log n)$ on n és la mida del vector:

```

int dicotomica(vector<int>& v, int e, int d, int x) {
    if (e > d) return -1;
    int m = (d + e)/2;
    if (v[m] < x) return dicotomica(v, m + 1, d, x);
    if (v[m] > x) return dicotomica(v, e, m - 1, x);
    return m;
}

```

Quin cost tindria aquest mateix algorisme si tinguéssim la mala pata d'oblidar-nos l'& en el pas de paràmetres? Resposta: $T(n) = \Theta(\quad)$.

Problema 2: Backtracking

(2 punts)

Considera el següent problema: donats n nombres enters positius a_0, a_1, \dots, a_{n-1} i dos altres enters l i u tals que $0 \leq l \leq u \leq \sum_{i=0}^{n-1} a_i$, es tracta de trobar totes les possibles solucions $(x_0, x_1, \dots, x_{n-1}) \in \{0, 1\}^n$ de la doble inequació:

$$l \leq a_0 x_0 + a_1 x_1 + \dots + a_{n-1} x_{n-1} \leq u$$

(a) (1 punt) Completa el programa següent perquè resolgui el problema anterior:

```

int n, l, u;
vector<int> a;

void sols(int k, int st, int sr, vector<bool>& x) {
    if (k == n) {
        for (int i = 0; i < n; ++i) cout << x[i];
        cout << endl;
    } else {
        if (  ) {
            x[k] = 0; sols(k+1, st, sr - a[k], x);
        }
        if (  ) {
            x[k] = 1; sols(k+1, st + a[k], sr - a[k], x);
        } }
}

int main() {
    cin >> n >> l >> u;
    a = vector<int>(n);
    for (int i = 0; i < n; ++i) cin >> a[i];

    int s = 0;
    for (int i = 0; i < n; ++i) s += a[i];

    vector<bool> x(n);
    sols(  ,  ,  , x);
}

```

(b) (1 punt) El programa de l'apartat anterior escriu les solucions en ordre lexicogràfic creixent. És a dir, per a l'entrada $n = 3, l = 3, u = 5$ i $a_i = i$ ($1 \leq i \leq 3$), es produeix la sortida:

```

001
011
101
110

```

Explica com modificar la funció *sols* perquè s'escriguin les solucions en ordre lexicogràfic decreixent, és a dir en l'ordre invers de l'anterior.

Problema 3: Segur que tinc sort

(3 punts)

Donat que ningú va entendre mai de què servia el botó "Segur que tinc sort", Google vol tornar a implementar-lo per veure si ara ... té més sort. Es vol que quan un usuari piqui el botó "Segur que tinc sort", el sistema retorni una pàgina escollida a l'atzar d'entre totes les pàgines indexades.

Per a fer-ho, cal disposar d'una estructura de dades que emmagatzemi un conjunt d'URLs (adreces de pàgines) tot donant suport a les operacions següents:

- Crear una estructura buida sense URLs.

- Inserir una URL al conjunt (sense repetits).
- Esborrar una URL del conjunt (no passa res si no hi era).
- Retornar una URL del conjunt triada a l'atzar amb probabilitat uniforme. És a dir, la probabilitat de retornar qualsevol URL del conjunt és exactament la mateixa que la de retornar qualsevol altra, i cada crida a aquesta funció retorna una nova URL a l'atzar. Com a precondition podem assumir que el conjunt no és buit.

Concretament, aquesta estructura de dades tindrà la interfície següent:

```
class UrlSet {  
public:  
    UrlSet ();  
    void insert_url (string url);  
    void remove_url (string url);  
    string random_url ();  
};
```

Un sistema d'escaneig automàtic de pàgines (el *crawler*, del qual no us n'heu d'encarregar) serà el responsable de mantenir l'estructura al dia: quan trobi una nova pàgina l'afegirà al conjunt, i quan trobi que una pàgina ja no existeix l'esborrarà del conjunt.

Tenint en compte que l'operació *random_url* s'aplicarà molt més freqüentment que les altres, descriuiu com implementar aquesta estructura de dades de forma que cada operació sigui el més eficient possible. Analitzeu el cost en espai de l'estructura de dades i el cost en temps de cada operació.

Nota: Supposeu que la funció `int randint(int a, int b)`; retorna nombres aleatoris entre *a* i *b* en cost constant. A més, us recomanem utilitzar tipus estàndards de la STL per descriure la vostra solució (però no és imprescindible). Fixeu-vos que no es demana una implementació completa. Penseu que aquest és un problema de disseny i per tant la millor solució no és única i depèn del criteri que hagueu decidit optimitzar (cas pitjor, cas mitjà, etc.).

Nota: Responen dins l'espai indicat a la pàgina del darrera. Feu un esquema de la vostra solució.

Problema 4: El creuer dels records

(2 punts)

Cinquanta anys més tard, la colla d'en Roy i en Johnny s'ha retrobat a les xarxes socials i han decidit d'anar a fer un creuer pel Mediterrani. Com que, entre tots, conserven una bona pila de records de quan eren joves, han fet una llista única de tot el que tenen i discuteixen via xat què en poden fer de tot això:

ROY: Si ens ho hem d'endur tot, això semblarà un *mercadillo*, no creus? A més, aquella foto on feia el pi no la vull veure ni en pintura.

JOHNNY: Ja ho entenc, ja. Mira, tinc una proposta: cadascú farà una comanda on digui quins objectes vol que es portin i quins no.

ROY: Bona idea, i de cada comanda intentarem que es compleixi almenys un desig: o que es porti un objecte que aquella persona vol, o que no se'n porti un dels que no vol. Si passa

això, ja es podrà donar per satisfet, no? Però necessitem algú que, a partir de les comandes, trobi una solució.

STEFFY: Ep, nois, que no teniu encara aquella llàntia màgica?

ROY: És clar, me n'oblidava, li treuré la pols i us diré què m'ha dit... —Al cap d'una estona—. Caram, el geni diu que només pot dir-nos si hi ha almenys **dues** solucions, és a dir, dos conjunts d'objectes que satisfacin a tothom. Però el que volíem era saber si n'hi havia almenys una!

STEFFY: Està clar que el geni ens vol posar a prova: el seu problema és NP-complet i només hem de trobar una transformació... Ja ho tinc! Crec que la solució és afegir un objecte i una comanda "fantasmes"!

JOHNNY: Mireu, amb els fantasmes i els genis jo em perdo: ho deixo a les vostres mans i ja m'avisareu, eh?

(a) (1 punt) Detalleu la reducció que ha trobat l'Steffy al problema del geni. Podeu suposar que hi ha k persones i , per tant, k comandes C_1, \dots, C_k . A més, tenim n objectes x_1, \dots, x_n , de manera que cada comanda C_i és un tuple amb objectes que la persona i vol i d'altres que no. Si, per exemple, la persona i vol els objectes x_1, x_5 però no vol x_2 , escrivim $C_i = (x_1, \neg x_2, x_5)$.

(b) (1 punt) Demostreu que el problema que volen resoldre, que anomenarem CNF-SAT, és a la classe NP.

Problema 5: Algorisme de Prim (competència transversal)

(1 punt)

Recordeu (de la pràctica de la competència transversal) que l'algorisme de Prim serveix per trobar un arbre d'expansió mínim en un graf connex amb pesos a les arestes.

(a) (0.5 punts) Doneu la definició d'arbre d'expansió mínim:

(b) (0.5 punts) Completeu el següent codi que implementa l'algorisme de Prim. La funció retorna un vector que, per cada vèrtex $u \in [0, \dots, n-1]$, apunta al pare de u dins l'arbre trobat (amb l'arrel 0 apuntant-se a si mateixa). Podeu suposar que el graf donat és connex.

```
typedef vector< pair<double, int> > WGraph;
typedef pair< double, pair<int, int> > WEdge;
```

```
vector<int> Prim(const WGraph& G) {
    vector<bool> usat(G.size(), false);
    vector<int> pare(G.size());
    priority_queue< WEdge> Q;
    Q.push(WEdge(0.0, pair<int, int>(0, 0)));
    while (not Q.empty()) {
        int u = Q.top().second.first;
        int v = Q.top().second.second;
        Q.pop();
```

```
    if (  ) {  
        usat[v] = true;  
        pare[v] = u;  
        for (int i = 0; i < G[v].size (); ++i) {  
            double p = G[v][i]. first ;  
            int w = G[v][i].second;  
              
        } } }  
    return pare ;  
}
```

Examen Final EDA

Duració: 3 hores

20/6/2014

Problema 1: Preguntes curtes (que ja van sortir al parcial)

(2 punts)

1. Sigui $f(n) = n \log(\cos(n\pi) + 4)$. Llavors $f(n) = \Theta(\quad)$.
2. Quin és l'últim dígit de 7^{1024} escrit en decimal? Resposta: \quad .
3. Quina és la **recurrència** que expressa el cost de l'algorisme de Strassen per multiplicar matrius $n \times n$. Resposta $T(n) = \quad$.
4. Un algorisme pot rebre els 2^n vectors de n bits com entrada. En $2^n - 1$ d'aquestes entrades el cost de l'algorisme és $\Theta(n^2)$ i en l'entrada restant el seu cost és $\Theta(n^4)$. Per tant, el cost en el cas pitjor és $\Theta(n^4)$ i el cost en el cas millor és $\Theta(n^2)$. Quin és el cost en el cas mitjà quan l'entrada s'escull uniformement a l'atzar (i per tant cada entrada té probabilitat $1/2^n$)?
Resposta: $T(n) = \Theta(\quad)$.

Justificacions per les preguntes 1, 2 i 4 (per la pregunta 3 **no** cal justificació):

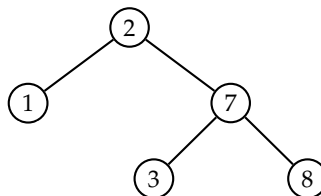
Problema 2: Arbres binaris de cerca i AVLs

(3 punts)

Per aquest problema feu servir la representació següent d'ABCs:

```
struct node {
    Elem x; // Informació en el node
    node* fe; // Punter al fill esquerre
    node* fd; // Punter al fill dret
    node(Elem x, node* fe, node* fd) : x(x), fe(fe), fd(fd) {}
    ~node() { delete fe; delete fd; }
};
typedef node* abc; // Un ABC és un punter a l'arrel (nul si és buit)
```

- (a) (1.0 punts) Implementeu en C++ una funció `abc refer(const vector<Elem> &v)` que reconstrueix un arbre binari de cerca a partir del seu recorregut en preordre v . Per exemple, amb l'entrada $v = [2 \ 1 \ 7 \ 3 \ 8]$ s'ha de produir l'arbre següent:



Podeu assumir que v és el preordre correcte d'algun ABC. Si us cal, implementeu una funció auxiliar.

- (b) (1.0 punts) Implementeu una funció `abc munta_AVL(const vector<Elem> &v)` en C++ que reconstrueix un ABC que satisfà la propietat d'AVL a partir d'un vector ordenat d'elements v . Si us cal, implementeu una funció auxiliar.

(c) (1.0 punts) Si afegim un camp $n.alc$ que denota l'alçada del node n en l'ABC on es troba (amb les fulles a alçada 0 i per tant els $NULL$ estarien a alçada -1), completeu el següent codi perquè el resultat indiqui si l'ABC T és un AVL:

```
bool es_AVL(abc T) {
    if (T == NULL) return true;
    else if (T->fe == NULL or T->fd == NULL) return (T->alc ≤ 1);
    else {
        ...
    }
}
```

Problema 3: Camins Hamiltonians en grafs torneig

(3 punts)

Representem grafs dirigits amb matrius d'adjacència:

```
typedef vector<vector<bool>> Graf;
```

Un *graf torneig* és un graf dirigit on per a cada parell de vèrtexs diferents hi ha exactament un arc, i sense arcs des de cap vèrtex cap a ell mateix.

a) (0.5 punts) Escriviu una funció en C++ que indiqui si un graf donat de n vèrtexs és o no un graf torneig en temps $\Theta(n^2)$ en el cas pitjor:

```
bool es_torneig (const Graf& G) {
    ...
}
```

b) (1.0 punts) Demostreu per inducció en el nombre de vèrtexs que tot graf torneig té un camí Hamiltonià, és a dir, un camí que visita tots els vèrtexs exactament una vegada (pista: mostreu que un nou vèrtex es pot inserir en un camí de $n - 1$ vèrtexs per obtenir un camí de n vèrtexs).

c) (1.0 punts) Valent-vos de la demostració anterior, doneu un algorisme que retorni un camí Hamiltonià d'un graf torneig (es valorarà més la claredat que l'eficiència):

```
// Pre: G és un graf torneig
// Post: el resultat és una llista de vèrtexs que forma un camí Hamiltonià a G
list<int> cami(const Graf &G) {
    ...
}
```

d) (0.5 punts) Analitzeu el cost de la vostra solució.

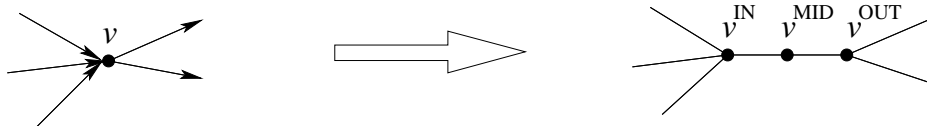
Problema 4: Reduccions

(1 punt)

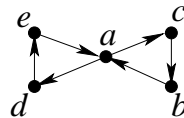
El problema dels cicles Hamiltonians en grafs dirigits és:

DHAM: Donat un graf dirigit, determinar si hi ha un cicle que passi per tots els vèrtexs exactament una vegada.

El problema dels cicles Hamiltonians en grafs no dirigits UHAM és el mateix però amb el graf d'entrada no dirigit. La Steffy sap que es pot reduir DHAM a UHAM simplement aplicant la següent transformació a cada vèrtex v del graf dirigit:



En Roy i en Johnny diuen que el vèrtex v^{MID} no cal: connectant $v^{\text{IN}} - v^{\text{OUT}}$ i ignorant v^{MID} és suficient. Però la Steffy replica: “Nois, mireu-vos aquesta entrada a DHAM i m’ho torneu a explicar”:



Què ha volgut dir la Steffy? Té raó? (sigueu concisos però precisos, si us plau).

Problema 5: Bellman-Ford (competència transversal)

(1 punt)

Recordeu (de la pràctica de la competència transversal) que l'algorisme de Bellman-Ford permet trobar camins de pes mínim en un graf dirigit amb pesos, fins i tot quan els pesos són negatius.

(a) (0.5 punts) Quin és el cost, en el cas pitjor, de l'algorisme de Bellman-Ford executat en un graf de $|V|$ vèrtexs i $|E|$ arestes? Resposta: $\Theta(\text{ })$

(b) (0.5 punts) La Steffy diu que l'afirmació del principi d'aquest problema no pot ser del tot correcta perquè altrament podríem trobar, de manera eficient, un cicle Hamiltonià en un graf dirigit $G = (V, E)$ simplement assignant pes -1 a cada aresta de G , escollint un vèrtex s qualsevol, i aplicant Bellman-Ford per mirar si el camí de pes mínim de s a algun dels vèrtexs que tenen una aresta cap a s té pes $-(|V| - 1)$. Com expliqueu la paradoxa? (o és que hem demostrat que $P = NP$?)

Examen Final EDA

Duració: 3 hores

16/1/2015

Problema d'anàlisi asimptòtica (que ja va sortir al parcial)

(2 punts)

(1 punt) Demostreu que $\sum_{i=0}^n i^3 = \Theta(n^4)$.

Ajut: Hi ha diverses maneres de fer-ho, una d'elles demostrant l' O i l' Ω per separat.

(1 punt) Siguin $f(n) = 2\sqrt{\log n}$ i $g(n) = n$. Asimptòticament, quina creix més ràpid? Demostreu-ho.

Problema 2: Arbres binaris de cerca i AVLs

(3 punts)

Amb la representació habitual d'arbres binaris de cerca (això també s'aplica als AVL) alguns algorismes resulten ineficients. Un exemple d'aquest fet és la funció *menors*(T, x) que donat un arbre T i una clau x , retorna el nombre d'elements de T que tenen clau $\leq x$.

Aquesta senzilla modificació de la representació

```
struct node {  
    node* esq;  
    node* dre;  
    Clau k;    // tipus genèric Clau  
    Valor v;   // tipus genèric Valor  
    int mida; // mida del subarbre arrelat en aquest node ***** NOU *****  
};  
typedef node* abc;
```

permet, entre altres coses, una implementació de la funció *menors*(T, x) amb cost $\Theta(h)$, on h és l'alçada de T (que seria $\Theta(\log n)$ en cas mitjà en un ABC aleatori i $\Theta(\log n)$ en cas pitjor en un AVL, si n és la mida de T).

(1,5 punts) Escriviu el codi C++ de la implementació eficient de la funció

```
int menors(abc T, const Clau& x)
```

(1,5 punts) Escriviu en C++ com s'ha de modificar el procediment *inserta*(T, k, v) que inserta un nou parell $\langle k, v \rangle$ en l'arbre T . La precondició d'aquest procediment assegura que la clau k no és a T . La implementació de *inserta* ha de tenir la mateixa eficiència que tindria si s'utilitzés la representació ordinària d'ABCs, però evidentment ha de mantenir actualitzat el camp *mida* de l'arbre T (noteu que **no** es demana mantenir la propietat d'AVL; ni es diu que l'arbre d'entrada T la tingui).

```
// Precondició: T no conté k  
void inserta (abc& T, const Clau& k, const Valor& v)
```

Problema 3: Grafs i NP-complets**(3 punts)**

Un estudiant d'EDA s'està preparant per a l'examen de laboratori i s'encalla en un problema de la llista "Graph Algorithms" del Jutge. Després de rumiar-hi una mica, veu que el que cal és, essencialment, donat un graf (no dirigit, representat amb llistes d'adjacència), determinar si és 2-colorable; és a dir, si és possible assignar un color *red* o *blue* a cada vèrtex del graf, de manera que vèrtexs adjacents no estiguin pintats amb el mateix color.

Malauradament, el Jutge respon Time Limit Exceeded a tots els enviaments que fa. Arribat a aquest punt, l'estudiant decideix buscar per Internet algun codi més eficient, i troba el següent penjat en un fòrum:

```
bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
                vector<bool>& marked, bool is_red) {
    col[u] = is_red ;
    marked[u] = true;
    for (int i = 0; i < g[u].size (); ++i) {
        int v = g[u][i];
        if (not marked[v]) {
            if (not two_col_aux(g, v, col, marked, not is_red )) return false;
        }
    }
    return true;
}

bool two_colourable (const vector<vector<int>>& g) {
    int n = g.size ();
    vector<bool> marked(n, false);
    vector<bool> col(n);
    for (int u = 0; u < n; ++u) {
        if (not marked[u]) {
            if (not two_col_aux(g, u, col, marked, false)) return false;
        }
    }
    return true;
}
```

Ara s'espera que aquest codi serà prou eficient, perquè després d'una mica d'anàlisi veu que triga $O(|V| + |E|)$. L'envia al Jutge, i resulta que ... Wrong Answer!

(2 punts) Escriviu una versió correcta de *two_col_aux* de manera que *two_colorable*, ara sí, determini si un graf és 2-colorable en temps $O(|V| + |E|)$.

```
bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
                vector<bool>& marked, bool is_red)
```

(1 punt) Finalment l'estudiant troba com arreglar el codi i que el Jutge l'hi accepti. Mosquejat, continua llegint el fòrum. L'autor del codi erroni afirma que el seu programa es pot estendre fàcilment al problema de 3-colorabilitat amb la mateixa complexitat asimptòtica. Us ho creieu? Per què?

Problema 4: Cultura algorísmica**(2 punts)**

Donat un graf dirigit on els arcs tenen costos que poden ser negatius, cal detectar si el graf té un o més cicles de cost total negatiu (no cal trobar cap d'aquests cicles, només dir si n'hi ha).

(0,5 punts) Doneu el nom, segons la literatura informàtica, d'un algorisme famós que permeti resoldre aquest problema fent-ne una petita adaptació.

(0,5 punts) Expliqueu breument, però amb prou detall per poder implementar-lo, com funciona aquest algorisme famós.

(0,5 punts) Quin cost té aquest algorisme? Expresseu-lo en funció del nombre de vèrtexs $|V|$ i del nombre d'arcs $|E|$.

(0,5 punts) Expliqueu com s'hauria d'adaptar l'algorisme general a aquest problema en particular (on només es demana detectar si hi ha cicles negatius).

Examen Final EDA**Duració: 3 hores****12/6/2015****Una del parcial, una de la competència transversal, i una de NP****(2 punts)**(1 punt) Determineu si són iguals ($=$) o diferents (\neq) i demostreu-ho:

$$\Theta(3^{\log_2(n)}) \square \Theta(3^{\log_4(n)}).$$

(0,5 punts) Per a què serveix l'algorisme que heu treballat a l'exercici de la competència transversal? Escolliu només una resposta (no cal justificació):

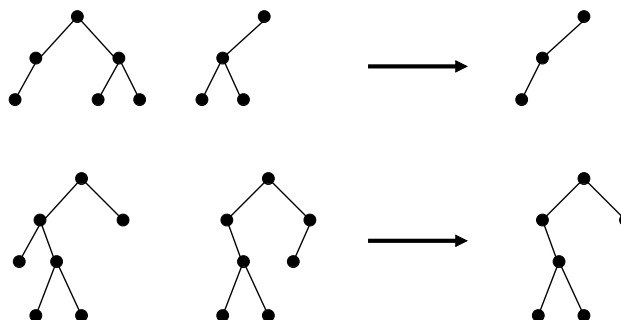
- Per a calcular l'arbre filogenètic més probable.
- Per a calcular el preu òptim de l'opció de compra d'un producte financer.
- Per a fer cabre les 9 simfonies de Beethoven en un sol CD.
- Per a resoldre recurrències que no tenen la forma dels teoremes mestre.

(0,5 punts) El problema SUMASUBCONJUNT és: Donada una llista de nombres naturals a_1, \dots, a_m i un objectiu b , tots escrits en decimal, determinar si existeix un subconjunt $S \subseteq \{1, \dots, m\}$ tal que $\sum_{i \in S} a_i = b$. Si d denota el nombre de dígit del màxim nombre natural de l'entrada, escolliu l'única afirmació que és correcta (no cal justificació):

- El problema és NP-complet i per tant no admet cap algorisme que tingui cost $O(m \cdot d \cdot 10^d)$ a menys que $P = NP$.
- El problema és NP-complet i per tant no admet cap algorisme que tingui cost polinòmic en m i d a menys que $P = NP$.
- El problema és NP-complet però malgrat tot i ha un algorisme conegut de cost polinòmic en m i d .
- El problema es pot resoldre en temps $O(d \cdot m \cdot \log m)$, i per tant temps polinòmic en la mida de l'entrada, així: primer s'inclouen els m nombres en un max-heap; després es van extraient d'un en un fins que, o bé ens passem de b , o bé s'acabin els nombres sense arribar a b , o bé els nombres extrets sumen exactament b .

Punters**(2 punts)**

Donats dos arbres binaris, crear un nou arbre binari que sigui la seva intersecció (i deixar els arbres donats com estaven). Per exemple:



(1.5 punts) Fent servir el tipus i la capçalera que es donen, resolcu aquest problema.

```
struct Node {    // els nodes no guarden dades; només punters als fills
    Node* esq;    // punter al fill esquerre
    Node* dre;    // punter al fill dret
};
```

```
typedef Node* Arbre;
```

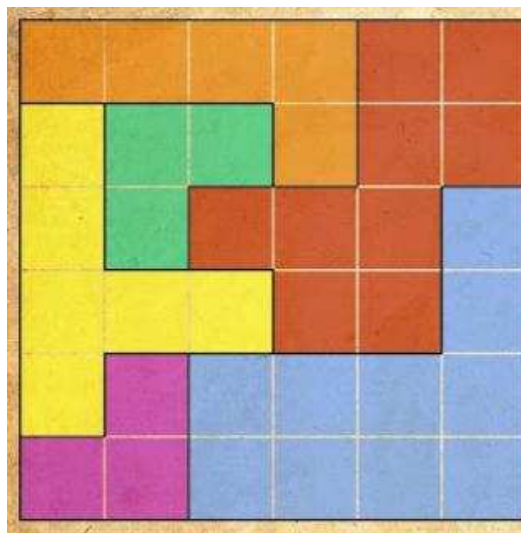
```
Arbre interseccio (Arbre a1, Arbre a2) {
    ...
}
```

(0.5 punts) Quin és el cost, en cas pitjor, en funció de les mides n_1 i n_2 de a_1 i a_2 ?

Backtracking fill-in-the-gaps

(2 punts)

Cebes és un trencaclosques³ de lògica en el qual s'ens dona una matriu quadrada $n \times n$ dividida en n regions, cadascuna pintada d'un color diferent.



L'objectiu és plantar-hi n cebes de manera que

- cada fila conté exactament una ceba
- cada columna conté exactament una ceba
- cada regió conté exactament una ceba
- no hi ha dues cebes adjacents en horitzontal, vertical o diagonal, dit d'una altra manera, totes les caselles envoltant una ceba estan lliures.

Fent servir les següents definicions:

```
struct Cell {
    int color;    // el color (de 1 a n) de la seva regio
    bool has_onion; // indica si hi ha una ceba plantada o no
```

³El trencaclosques original es diu *Alberi* (obre en Italià) i les apps (per iOS i Android) per aquest joc van ser molt populars fa uns anys.

```
}
```

```
typedef vector<vector<Cell>> > OnionBoard;
```

es demana que completeu el codi del següent algorisme de *backtracking* per trobar una solució, o determinar que no n'hi ha. Si n'hi ha més d'una, qualsevol solució serveix.

```
class Onion {
    int n;
    bool solution_found;
    OnionBoard board;           // solució parcial en curs
    vector<int> col;             // col[i] = columna de la ceba de la fila i-essima
    vector<bool> onion_in_col;   // onion_in_col[i] = hi ha ceba a la columna i
    vector<bool> onion_in_region; // onion_in_region[i] = hi ha ceba a la regio i
```

```
public:
```

```
    Onion(const OnionBoard& initial_board) {
        board = initial_board;
        n = board.size();
        col = vector<int>(n);
        onion_in_col = vector<bool>(n, false);
        onion_in_region = vector<bool>(n, false);
        solution_found = false;
        backtrack(0);
    }
```

```
void backtrack(int k) {
    if (k == n) { solution_found = true; return; }
    for (int j = 0; j < n and not solution_found; ++j) {
```

```
        int cell_col = 
        if (not onion_in_col[j] and not onion_in_region[cell_col]
            and not onion_in_neighborhood(k, j)) {
            col[k] = j;
```

```
            backtrack(k+1);
```

```
        } } }
```

```
// indica si la solucio parcial board te cebes al veinatge de la casella (i,j)
// no cal que la implementeu.
```

```
bool onion_in_neighborhood(int i, int j) const;
```


Piulades**(4 punts)**

Una xarxa social tipus Twitter té diferents usuaris. Aquests mantenen una relació de seguiment d'altres usuaris (per exemple, en Salvador segueix l'Anna, la Ivet i en Leo Messi, però en Leo Messi no segueix ningú). Els usuaris poden afegir o eliminar altres usuaris de la seva llista d'usuaris seguits. A més, en qualsevol moment, un usuari pot piular un missatge. També, en qualsevol moment, un usuari pot consultar les k piulades més recents de tots els usuaris que segueix. (És a dir, si segueix n usuaris, en total vol (com a molt) k piulades, no nk piulades.) Les dimensions d'aquest sistema són les que hom podria imaginar:

- El nombre de usuaris és enorme; el de piulades encara més.
- El nombre d'usuaris que un usuari segueix és molt variable, però no enorme.
- Alguns usuaris són seguits per molts altres usuaris.
- El nombre de piulades per usuari és molt variable.
- El valor de k és relativament petit, i tant podria ser 10 com 1000.

Els usuaris es representen amb un struct *Usuari* que conté les seves dades personals (incloent un *string* nom que l'identifica de forma única) i les piulades es representen amb un struct tupla *Piulada* que conté el missatge i l'hora d'emissió:

```
struct Usuari {
    ...                // dades personals
    string nom;        // no hi ha dos usuaris amb el mateix nom
};

struct Piulada {
    string missatge;    // text del missatge
    int hora_emissio;   // nombre de milisegons des d'un moment
};
```

Expliqueu quines estructures de dades i algorismes utilitzaríeu per guardar la informació descrita i implementeu tant eficient com sigui possible les operacions d'afegir i eliminar el seguiment d'un usuari, de piular un missatge, i d'obtenir les k piulades més recents dels usuaris seguits per un usuari donat. No cal que expliqueu com es donen d'alta/de baixa els usuaris. Analitzeu el cost de les operacions.

La resposta esperada és oberta però, per concreció, definiu l'estructura de dades *Xarxa* i descriviu (sense donar codi) com implementaríeu les operacions:

```
struct Xarxa { ... };

// u1 passa a seguir a u2 (si no ho feia); els usuaris existeixen.
void afegir_seguiment (Xarxa& x, string u1, string u2);

// u1 deixa de seguir a u2 (si ho feia); els usuaris existeixen.
void treure_seguiment (Xarxa& x, string u1, string u2);

// u piula el missatge m; l'usuari existeix.
void piular_missatge (Xarxa& x, string u, Piulada m);

// Retorna les k piulades més recents dels usuaris seguits per u; l'usuari existeix.
list<Piulada> k_piulades_mes_recents_dels_seguits (const Xarxa& x, string u, int k);
```

Examen Final EDA**Duració: 3 hores****07/01/2016****Problema 1****(3 pts.)**

Responen les següents preguntes:

(a) (1 pt.) Considereu els següents problemes decisionals:

- SAT: donada una fórmula proposicional, determinar si és satisfactible.
- COL: donats un graf $G = (V, E)$ i un nombre natural k , determinar si es pot pintar G amb k colors de forma que les arestes tinguin els extrems pintats amb colors diferents.
- SOR: donada una taula de n enters diferents, determinar si està ordenada de forma creixent.

Per cada afirmació a continuació, marqueu amb una 'X' la columna corresponent, segons si l'afirmació és certa, falsa, o és un problema obert i no se sap encara si és certa o falsa. No cal justificació.

	Cert	Fals	Obert
SOR és a la classe P			
SOR és a la classe NP			
SOR és NP-difícil			
SAT és a la classe P			
SAT és a la classe NP			
SAT és NP-difícil			
Es pot reduir polinòmicament SOR a COL			
Es pot reduir polinòmicament COL a SOR			
No es pot reduir polinòmicament SAT a SOR			
Es pot reduir polinòmicament COL a SOR, i no es pot reduir polinòmicament SAT a SOR			

(b) (1 pt.) Donats un vector d'enters v i un enter x , la funció

```

int posicio (const vector<int>& v, int x) {
    int n = v.size ();
    for (int i = 0; i < n; ++i)
        if (v[i] == x) return i;
    return -1;
}

```

retorna la primera posició de v que conté x , o -1 si no n'hi ha cap.

Considereu una distribució de probabilitat sobre els paràmetres d'entrada tal que la probabilitat que x sigui l'element $v[i]$ és $\frac{1}{n}$ per a $0 \leq i < n$.

Responen: el cost en temps de *posicio* en el cas mig en funció de n és $\Theta(\text{ } \boxed{\text{ }} \text{ })$.

Justificació:

(c) (1 pt.) Què calcula l'algorisme de Floyd-Warshall? Quin és el seu cost en temps i espai en el cas pitjor? (no cal justificació)

Problema 2

(2 pts.)

Sigui V un `vector<int>` amb n enters diferents. Considereu aquest codi:

```
set<int> S;  
for (int i = 0; i < n; ++i) {  
    S.insert(V[i]);  
    cout << *S.begin() << ' '  
}
```

- (a) (0.4 pts.) Què escriu el codi si $n = 6$ i els elements de V són 42 100 23 12 20 24 (en aquest ordre)?

- (b) (0.4 pts.) Doneu una explicació d'alt nivell sobre quins valors escriu el codi amb una V qualsevol amb n enters diferents.

- (c) (0.4 pts.) Raoneu quin cost asimptòtic en temps té el codi en el cas pitjor en funció de n .

Pista: Podeu fer servir la fórmula d'Stirling: $\log(n!) = \Theta(n \log n)$.

- (d) (0.4 pts.) Si els elements de V estiguessin repetits, el codi podria calcular alguna cosa diferent a la mencionada a l'apartat (b)? Si la resposta és afirmativa, mostreu un exemple. Si la resposta és negativa, expliqueu per què.

- (e) (0.4 pts.) Escriviu amb tot detall un codi alternatiu al codi donat, que calculi el mateix amb un vector V qualsevol amb n enters diferents ($n \geq 1$), però amb cost asimptòtic en temps estrictament menor al calculat a l'apartat (c). Raoneu quin és aquest cost asimptòtic.

Problema 3**(2 pts.)**

Considerem el problema de, donat un graf dirigit acíclic $G = (V, E)$, generar totes les seves ordenacions topològiques. El següent programa el resol (assumint $V = \{0, 1, \dots, n - 1\}$):

```

#include <iostream>
#include <vector>
using namespace std;

typedef vector<int> AdjList;
typedef vector<AdjList> Graph;

// implementation not given here
Graph read_graph ();
void print_solution (const vector<int>& sol);

bool ok(const Graph& G, const vector<int>& sol) {
    int n = sol.size ();
    vector<bool> mar(n, false);
    for (int i = 0; i < n; ++i) {
        int x = sol[i];
        if (mar[x]) return false;
        mar[x] = true;
        for (int y : G[x])
            for (int j = 0; j < i; ++j)
                if (sol[j] == y) return false;
    }
    return true;
}

void top_sorts_rec (int k, const Graph& G, vector<int>& sol) {
    int n = sol.size ();
    if (k == n) {
        if (ok(G, sol))
            print_solution (sol);
    }
    else
        for (int x = 0; x < n; ++x) {
            sol[k] = x;
            top_sorts_rec (k+1, G, sol);
        }
}

void top_sorts (const Graph& G, vector<int>& sol) {
    top_sorts_rec (0, G, sol);
}

int main() {
    Graph G = read_graph ();
    vector<int> sol(G.size ());
    top_sorts (G, sol);
}

```

Tanmateix, és massa lent. Reescriu la funció `top_sorts` per resoldre el problema més eficientment, sense canviar el `main`. Implementeu també les funcions auxiliars que useu, excepte `print_solution` i les funcions de la llibreria estàndard de C++.

Problema 4**(3 pts.)**

Sigui G la matriu d'adjacència d'un graf dirigit amb n vèrtexs. La *matriu de clausura* de (el graf representat per) G , que representarem com G^* , és una matriu $n \times n$ definida de la manera següent: donats u, v tals que $0 \leq u, v < n$, el coeficient a la fila u i columna v és

$$G_{uv}^* = \begin{cases} 1 & \text{si hi ha un camí (potser buit) de } u \text{ a } v \text{ en el graf} \\ 0 & \text{altrament} \end{cases}$$

on els vèrtexs del graf s'identifiquen amb els enters de 0 a $n - 1$ com és habitual.

(a) (1 pt.) Descriviu a alt nivell com implementar una funció

```
vector<vector<int>> clausura(const vector<vector<int>> & G);
```

que, donada la matriu d'adjacència G d'un graf dirigit, retorni la seva clausura. Analitzeu-ne el cost asimptòtic en temps en el cas pitjor.

(b) (1 pt.) Representem per I la matriu identitat $n \times n$, és a dir, la matriu on tots els coeficients són 0 llevat dels de la diagonal, que són 1. Demostreu que, per tot $k \geq 0$, hi ha un camí en el graf de u a v amb com a molt k arestes si i només si el coeficient a la fila u columna v de la matriu $(I + G)^k$ (la matriu resultant de multiplicar k matrius $I + G$) és diferent de 0, és a dir, $(I + G)_{uv}^k \neq 0$.

- (c) (1 pt.) Descriuiu a alt nivell un algorisme per, donada la matriu d'adjacència G d'un graf dirigit, calcular-ne la seva clausura G^* en temps estrictament millor que $\Theta(n^3)$ en el cas pitjor. Doneu-ne el cost en el cas pitjor i justifiqueu-lo. Assumiu que el cost de les operacions aritmètiques amb enters és constant.

Pista: si hi ha camí de u a v , almenys n'hi ha un amb com a molt $n - 1$ arestes.

Examen Final EDA

Duració: 3 hores

08/06/2016

Problema 1

(1.5 pts.)

Responen les següents preguntes:

- (a) (0.9 pt.) El teorema mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma $T(n) = aT(n/b) + \Theta(n^k)$ amb $a > 0$, $b > 1$ i $k \geq 0$, llavors, fent $\alpha = \log_b a$,

$$T(n) = \begin{cases} \boxed{} & \text{si } \alpha < k, \\ \boxed{} & \text{si } \alpha = k, \\ \boxed{} & \text{si } \alpha > k. \end{cases}$$

- (b) (0.3 pt.) La k més petita tal que el següent enunciat és cert:

“Per a tot graf dirigit $G = (V, E)$, es compleix $|E| = O(|V|^k)$ ”

és .

- (c) (0.3 pt.) La classe `stack<T>` de la STL té dues funcions membre `top`:

```
T& top();
const T& top() const;
```

En canvi, la classe `priority_queue<T>` només té una funció `top`:

```
const T& top() const;
```

Quin inconvenient hi ha en què la classe `priority_queue<T>` tingui una funció `T& top()`?

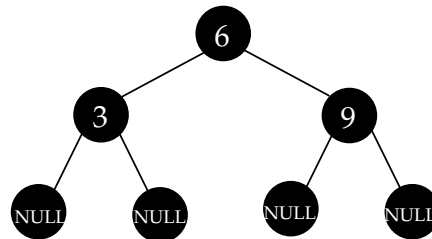
Problema 2

(1.5 pts.)

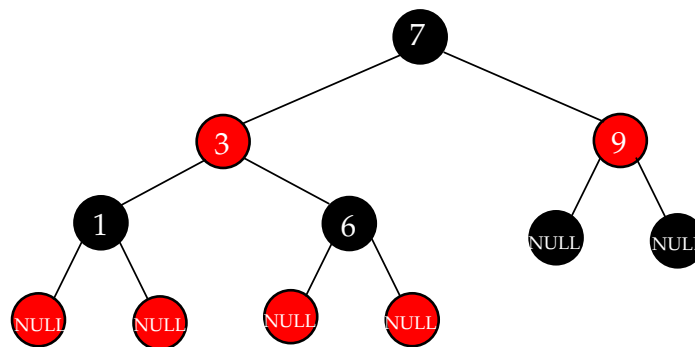
En aquest exercici, els nodes sombrejats representen pintats de color vermell.

Les respostes invàlides resten amb la mateixa puntuació que sumen les correctes.

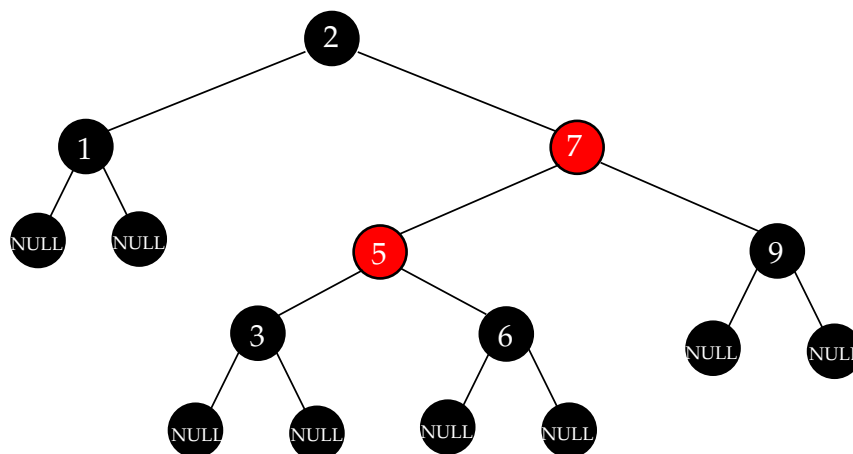
(a) El següent arbre binari és un arbre vermell-negre (responeu **Sí** o **No**):



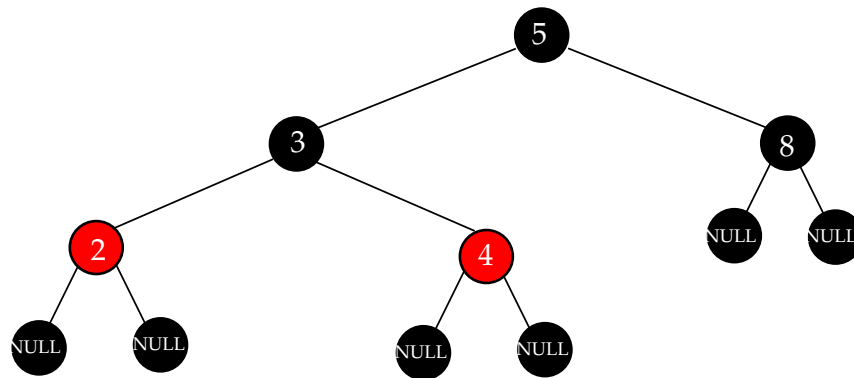
(b) El següent arbre binari és un arbre vermell-negre (responeu **Sí** o **No**):



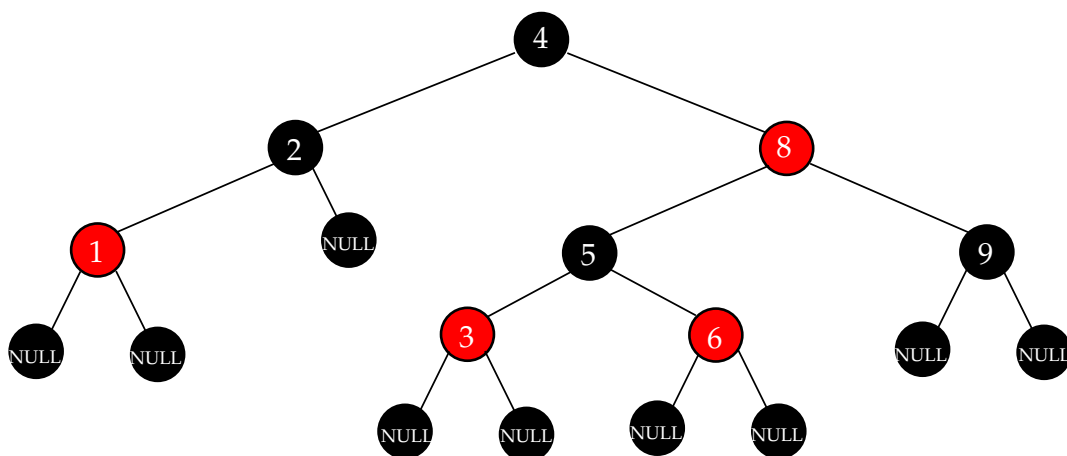
(c) El següent arbre binari és un arbre vermell-negre (responeu **Sí** o **No**):



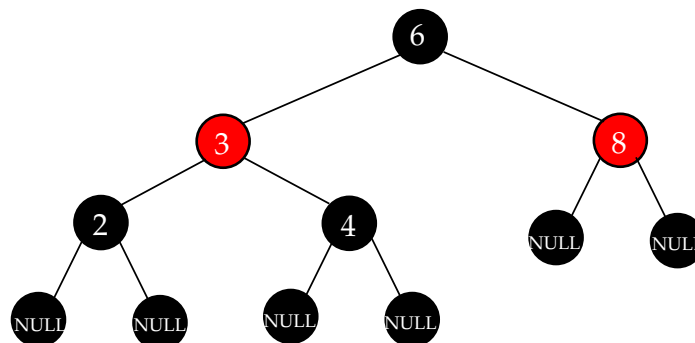
(d) El següent arbre binari és un arbre vermell-negre (responeu **Sí** o **No**):



(e) El següent arbre binari és un arbre vermell-negre (responen **Sí** o **No**):



(f) El següent arbre binari és un arbre vermell-negre (responen **Sí** o **No**):



Problema 3

(2 pts.)

Sigui $s \in \mathbb{R}$ tal que $0 < s < 1$.

Considerem la següent variant de la cerca dicotòmica que, donats:

- un element x ,

- un vector v ordenat de forma no-decreixent, i
- dues posicions l i r ,

retorna una posició entre l i r , ambdues incloses, en què x apareix a v (o -1 en cas que no n'hi hagi cap):

double s ; // $0 < s < 1$

```
int posicio (double  $x$ , const vector<double>&  $v$ , int  $l$ , int  $r$ ) {  
    if ( $l > r$ ) return  $-1$ ;  
    int  $p = l + \text{int}(s * (r - l))$ ;  
    if ( $x < v[p]$ ) return posicio ( $x$ ,  $v$ ,  $l$ ,  $p - 1$ );  
    if ( $x > v[p]$ ) return posicio ( $x$ ,  $v$ ,  $p + 1$ ,  $r$ );  
    return  $p$ ;  
}
```

- (a) (0.75 pt.) Segons el valor del paràmetre s , doneu una recurrència que descrigui el cost en temps en el cas pitjor de *posicio* en funció de $n = r - l + 1$.

- (b) (0.5 pt.) Segons el valor de s , quan es dóna el cas pitjor?

- (c) (0.75 pt.) Segons el valor de s , resolueu la recurrència de la resposta de l'apartat (a) i doneu el cost asimptòtic en el cas pitjor.

Problema 4**(2 pts.)**

Un *deque*<*T*> (acrònim de *double-ended queue*, pronunciat: “dèk”) és un contenidor seqüencial d’objectes de tipus *T* que permet afegir i treure elements tant al principi com al final, mitjançant les següents funcions:

```
void push_front (const T& x);  
void pop_front ();
```

```
void push_back (const T& x);  
void pop_back ();
```

respectivament. A més, es pot recuperar l’element del principi amb les funcions:

```
T& front ();  
const T& front() const;
```

Totes aquestes funcions tenen cost en temps $\Theta(1)$.

- (a) (1 pt.) Un graf dirigit $G = (V, E)$ amb pesos és *binari* si per tota aresta $e \in E$, el seu pes és 0 o 1. Utilitzant un *deque*, implementeu en C++ una funció

```
int cost_minim(const vector<vector<pair<int,int>>>& G, int x, int y);
```

que, donats un graf binari G i dos vèrtexs x i y de G , retorni el cost mínim d'anar de x a y (o -1 si no es pot arribar de x a y).

Assumiu que els vèrtexs es representen amb enters entre 0 i $n - 1$, on n és el nombre de vèrtexs; i que el graf està representat amb llistes d'adjacències amb un `vector<vector<pair<int,int>>>`, on la primera component dels *pairs* representa el vèrtex successor i la segona component, el pes de l'aresta.

- (b) (0.5 pt.) Analitzeu el cost en temps de la vostra funció `cost_minim` en el cas pitjor en funció del nombre de vèrtexs $|V|$ i d'arestes $|E|$.

- (c) (0.5 pt.) Quin cost en temps en el cas pitjor tindria *cost_minim* si uséssim la versió de l'algorisme de Dijkstra que implementa la cua de prioritats amb un heap binari (com els vistos a classe)?

Problema 5**(3 pts.)**

El problema de DESIGUALTATS consisteix en, donats un interval d'enters i un conjunt de desigualtats (\neq) entre variables, determinar si hi ha una solució de totes les desigualtats amb valors en l'interval. Més formalment, donats:

- un interval (finit) $[l, u] \subset \mathbb{Z}$,
- un conjunt de variables V , i
- un conjunt D de desigualtats de la forma $x \neq y$, on $x, y \in V$,

cal determinar si hi ha $s : V \rightarrow \mathbb{Z}$ tal que:

- per tot $x \in V$ es té $s(x) \in [l, u]$, i
- per tot $x \neq y \in D$ es compleix $s(x) \neq s(y)$.

Considerarem que el conjunt de variables és de la forma $V = \{x_0, x_1, \dots, x_{n-1}\}$ per a una certa $n > 0$, i identificarem les variables amb enters entre 0 i $n - 1$. A més, representarem les entrades per al problema de DESIGUALTATS mitjançant el tipus `ent_DESIGUALTATS` definit així:

```
struct ent_DESIGUALTATS {
    int l, u, n;
    set<pair<int,int>> D; // cada pair (i,j) es una desigualtat  $x_i \neq x_j$ 
};
```

(a) (1 pt.) Completeu la següent implementació de la funció

bool *te_solucio* (**const** *ent_DESIGUALTATS*& *e*);

que resol el problema de DESIGUALTATS:

```
bool ok(const vector<int>& s, const set<pair<int,int>>& D) {
    for (auto d : D)
        if (  )
            return false;
    return true;
}
```

```
bool te_solucio (int k, vector<int>& s, const ent_DESIGUALTATS& e) {
    if (k == e.n) return ok(s, e.D);
    for (int v =  ; v <=  ; ++v) {
        
        if ( te_solucio (k+1, s, e)) return true;
    }
    return false;
}
```

```
bool te_solucio (const ent_DESIGUALTATS& e) {
    vector<int> s(e.n);
    return te_solucio (  , s, e);
}
```

(b) (0.75 pt.) Demostreu que el cost en temps en el cas pitjor de la implementació anterior de *te_solucio* és $\Omega((u - l + 1)^n)$.

(c) (0.75 pt.) Considerem el problema de COLOREJAT: donats un graf no dirigit G i un nombre natural $c > 0$, determinar si es poden pintar els vèrtexs de G amb c colors de forma que tota aresta tingui els extrems pintats amb colors diferents.

Suposem que representem les entrades per al problema de COLOREJAT amb el tipus *ent.COLOREJAT* definit així:

```
struct ent_COLOREJAT {  
    vector<vector<int>> G; // graf representat amb llistes d'adjacència  
    int c;  
};
```

Implementeu una reducció polinòmica de COLOREJAT a DESIGUALTATS:

```
ent_DESIGUALTATS reduccio(const ent_COLOREJAT& ec);
```

(d) (0.5 pt.) Veieu factible trobar un algorisme polinòmic per a DESIGUALTATS? Per què?

Examen Final EDA

Duració: 3 hores

12/01/2017

Problema 1

(2 pts.)

Ompliu els blancs següents de la forma més precisa possible:

- (a) (0.25 pts.) Un graf de n vèrtexs té $O(\quad)$ arestes.
- (b) (0.25 pts.) Un graf connex de n vèrtexs té $\Omega(\quad)$ arestes.
- (c) (0.25 pts.) Un graf complet de n vèrtexs té $\Omega(\quad)$ arestes.
- (d) (0.25 pts.) Un min-heap de n vèrtexs té $\Theta(\quad)$ fulles.
- (e) (0.25 pts.) Un arbre binari de cerca de n vèrtexs té alçada $\Omega(\quad)$.
- (f) (0.25 pts.) Un arbre binari de cerca de n vèrtexs té alçada $O(\quad)$.
- (g) (0.25 pts.) Un arbre AVL de n vèrtexs té alçada $\Omega(\quad)$.
- (h) (0.25 pts.) Un arbre AVL de n vèrtexs té alçada $O(\quad)$.

Problema 2

(3 pts.)

Sigui $G = (V, E)$ un graf dirigit amb pesos $\omega : E \rightarrow \mathbb{R}$. Volem resoldre el problema de, donat un vèrtex $s \in V$, calcular la distància de s a tots els vèrtexs del graf.

Recordem que:

- un *camí* és una seqüència de vèrtexs connectats consecutivament per arcs; és a dir, (u_0, u_1, \dots, u_k) tal que per tot $1 \leq i \leq k$, tenim $(u_{i-1}, u_i) \in E$.
- el *pes* d'un camí és la suma de pesos dels arcs que el formen:

$$\omega(u_0, u_1, \dots, u_k) = \sum_{i=1}^k \omega(u_{i-1}, u_i).$$

- la *distància* d'un vèrtex u a un vèrtex v és el pes del camí (anomenat *camí mínim*) amb pes mínim dels que surten de u i arriben a v , si existeix.
- (a) (0.2 pts.) Suposem que per tota aresta $e \in E$, tenim $\omega(e) = 1$; és a dir, tots els pesos són 1. Quin algorisme podem usar per resoldre eficientment el problema de les distàncies?

- (b) (0.2 pts.) Suposem que per tota aresta $e \in E$, tenim $\omega(e) \geq 0$; és a dir, tots els pesos són no negatius. Quin algorisme podem usar per resoldre eficientment el problema de les distàncies?

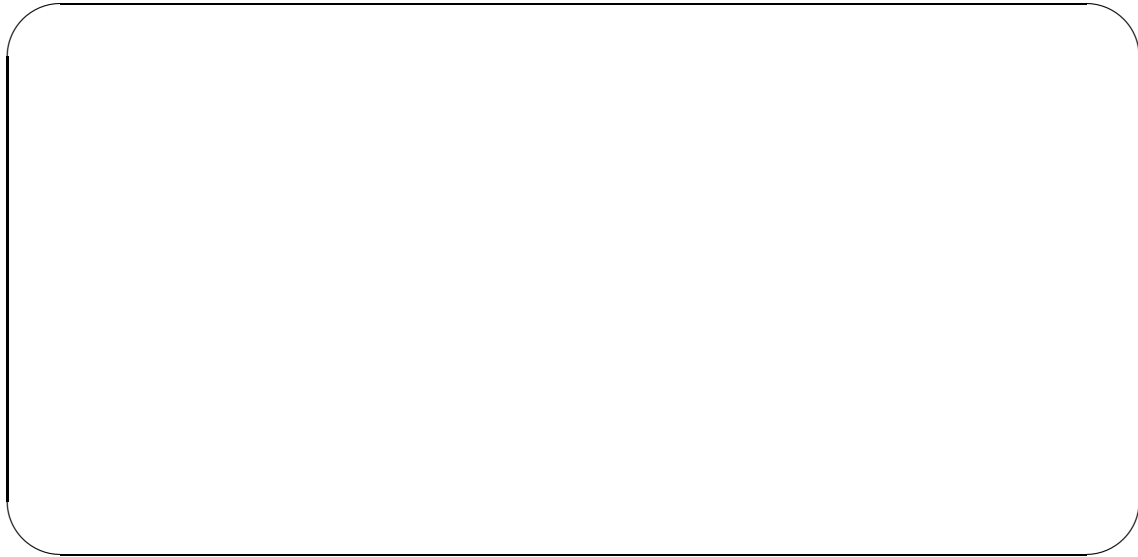
- (c) (0.2 pts.) Quin algorisme podem usar per resoldre eficientment el problema de les distàncies, si alguns pesos poden ser negatius?

- (d) (0.4 pts.) Quina condició sobre els cicles del graf cal que es compleixi per tal que, per tot parell de vèrtexs $u, v \in V$, existeixi un camí mínim de u a v ?

- (e) (1 pt.) Una funció $\pi : V \rightarrow \mathbb{R}$ és un *potencial* del graf si compleix que, per tota aresta $(u, v) \in E$, tenim $\pi(u) - \pi(v) \leq \omega(u, v)$. A més, es defineixen els *pesos reduïts* ω_π com $\omega_\pi(u, v) = \omega(u, v) - \pi(u) + \pi(v)$ per tot $(u, v) \in E$.

Demostreu que si c és un camí de u a v llavors $\omega_\pi(c) = \omega(c) - \pi(u) + \pi(v)$.

- (f) (1 pt.) Suposem que el graf té un potencial π . Llavors es pot demostrar que per tot parell de vèrtexs $u, v \in V$ hi ha un camí mínim amb pesos ω de u a v . Assumint aquest fet, expliqueu com usar el potencial π per calcular la distància d'un vèrtex donat s a tots els vèrtexs del graf amb un algorisme alternatiu al de l'apartat (c) quan els pesos poden ser negatius.

**Problema 3****(2 pts.)**

Definim un tipus *matrix* per a representar matrius quadrades de nombres reals. Considereu el següent programa:

```
typedef vector<vector<double>> matrix;

matrix aux(const matrix& A, const matrix& B) {
    int n = A.size ();
    matrix C(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                C[i][j] += A[i][k] * B[k][j];
    return C;
}

matrix mystery(const matrix& M) {
    int n = M.size ();
    matrix P(M), Q(M);
    for (int i = 1; i < n; ++i) {
        P = aux(P, M);
        Q = aux(Q, P);
    }
    return Q;
}
```

- (a) (0.5 pts.) Què calcula la funció *matrix* `mystery(const matrix& M)` en termes de la matriu *M*?

- (b) (0.5 pts.) Si M és una matriu $n \times n$, quin és el cost en el cas pitjor de la funció
- ```
matrix mystery(const matrix& M)
```
- en funció de  $n$ ? Justifiqueu la resposta.

- (c) (1 pt.) Implementeu en C++ una funció que calculi el mateix que la funció

```
matrix mystery(const matrix& M)
```

i que trigui temps  $\Theta(n^3 \log n)$  en el cas pitjor. Implementeu també les funcions auxiliars que useu, excepte *aux* i les funcions de la llibreria estàndard de C++. Justifiqueu que el cost de la vostra funció és com es demana.

**Problema 4****(3 pts.)**

El problema de GRAF HAMILTONIÀ consisteix en, donat un graf (no dirigit), decidir si és Hamiltonià, és a dir, si hi ha un cicle que passa per tots els seus vèrtexs una sola vegada. Se sap que GRAF HAMILTONIÀ és un problema NP-complet.

També se sap que, a partir de la demostració que un problema pertany a la classe NP, es pot derivar un algorisme de força bruta que el resol. La següent funció

**bool** *ham*(**const** **vector**<**vector**<**int**>>& *G*) **que**,

donat un graf *G* representat amb llistes d'adjacència, diu si *G* és Hamiltonià, implementa aquest algorisme en el cas de GRAF HAMILTONIÀ.

```
1 bool ham_rec(const vector<vector<int>>& G, int k, vector<int>& p) {
2 int n = G.size ();
3 if (k == n) {
4 vector<bool> mkd(n, false);
5 for (int u : p) {
6 if (mkd[u]) return false;
7 mkd[u] = true;
8 }
9 for (int k = 0; k < n; ++k) {
10 int u = p[k];
11 int v;
12 if (k < n - 1) v = p[k + 1];
13 else v = p[0];
14 if (find(G[u].begin (), G[u].end (), v) == G[u].end()) return false;
15 }
16 return true;
```

```

17 }
18 for (int v = 0; v < n; ++v) {
19 p[k] = v;
20 if (ham_rec(G, k+1, p)) return true;
21 }
22 return false;
23 }
24
25 bool ham(const vector<vector<int>>& G) {
26 int n = G.size ();
27 vector<int> p(n);
28 return ham_rec(G, 0, p);
29 }

```

Nota: La funció de la llibreria STL

*Iterator find ( Iterator first , Iterator last , int val );*

retorna un iterador al primer element en el rang  $[first, last)$  que és igual a *val*. Si tal element no existeix, la funció retorna *last*.

- (a) (0.5 pts.) Identifiqueu la variable en el programa anterior que representa el testimoni quan la funció *ham* retorna **true**.

- (b) (0.5 pts.) Identifiqueu el codi corresponent al verificador en el programa anterior. Per fer-ho, useu el números de línia del marge esquerre.

- (c) (1 pt.) Ompliu els blancs següents per tal que la funció *ham2* calculi el mateix que la funció *ham*, però més eficientment.

```

bool ham2_rec(const vector<vector<int>>& G, int k, int u, vector<int>& next) {
 int n = G.size ();
 if (== n)
 return find (G[u].begin (), G[u].end (),) != G[u].end();

 for (int v : G[u])
 if (next[v] ==) {
 next[u] = ;
 if (ham2_rec(G, k+1, v, next)) return true;
 next[u] = -1;
 }
 return false;
}

bool ham2(const vector<vector<int>>& G) {

```



```
int n = G.size ();
vector<int> next(n, -1);
return ham2_rec(G, , 0, next);
}
```

- (d) (0.5 pts.) Suposem que les llistes d'adjacència de la representació de  $G$  estan ordenades (per exemple, de forma creixent). Expliqueu com utilitzar això per fer més eficient la funció de l'apartat (c).

- (e) (0.5 pts.) Suposem que  $G$  és un graf **no connex**. Expliqueu com utilitzar això per fer més eficient la funció de l'apartat (c).

## Examen Final EDA

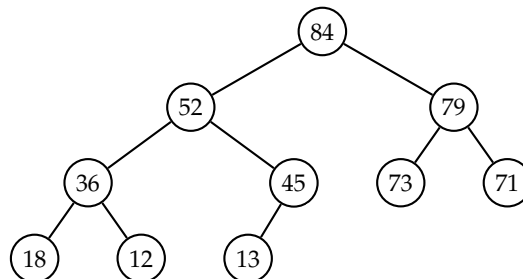
Duració: 3 hores

09/06/2017

## Problema 1

(2 pts.)

(a) (0.5 pts.) Donat el max-heap següent:



dibuixeu el max-heap resultant d'afegir 55 i d'esborrar l'element màxim (en aquest ordre). No cal justificar res.

(b) (0.5 pts.) Usant que  $P \subseteq NP$ , demostreu que  $P \subseteq \text{co-NP}$ .

(c) (0.5 pts.) Considereu la funció següent:

```

int f(const vector<int>& v, int i, int j) {
 // Precondition: $0 \leq i \leq j < v.size()$
 if (i == j) return v[i];
 int p = (j - i + 1)/3;
 int m1 = i+p;

```

```
int m2 = m1+p;
return f(v, i, m1) + f(v, m1+1, m2) + f(v, m2+1, j);
}
```

Calculeu el cost asimptòtic temporal de  $f$  en funció de  $n = j - i + 1$ .

(d) (0.5 pts.) Considereu la següent afirmació:

La funció  $n^n$  creix asimptòticament més ràpid que qualsevol altra funció.

És certa? Si és així, justifiqueu-ho. Altrament doneu-ne un contraexemple.

## Problema 2

(2 pts.)

Donats un vector  $A$  amb  $n$  enters diferents i un altre vector  $B$  amb  $m$  enters diferents, volem calcular un vector (d'enters diferents) que sigui la intersecció dels dos (és a dir, que contingui els elements comuns).

Per exemple, si  $A = (3, 1, 6, 0)$  i  $B = (4, 6, 1, 2, 7)$ , aleshores qualsevol dels dos vectors  $(1, 6)$ ,  $(6, 1)$  seria una resposta vàlida.

Suposem que  $A$  i  $B$  **no estan necessàriament** ordenats. Doneu una descripció a alt nivell de com implementaríeu una funció

**vector<int> intersection (const vector<int>& A, const vector<int>& B);**

que retorni la intersecció d' $A$  i  $B$  amb un cost temporal  $O(n + m)$  **en el cas mitjà**. Justifiqueu el cost.

### Problema 3

(3 pts.)

Donat un graf dirigit acíclic (DAG)  $G$ , el *nivell* dels seus vèrtexs es defineix inductivament de la forma següent:

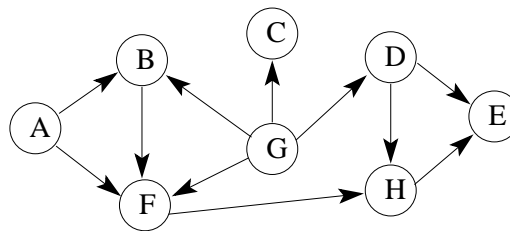
- si  $v$  és una arrel de  $G$  (un vèrtex sense predecessors) aleshores  $\text{nivell}(v) = 0$
- altrament,

$$\text{nivell}(v) = 1 + \max\{\text{nivell}(u) \mid u \text{ és un predecessor de } v\}$$

A més, la profunditat de  $G$  és el nivell més gran de qualsevol vèrtex:

$$\text{profunditat}(G) = \max\{\text{nivell}(v) \mid v \text{ vèrtex de } G\}$$

- (a) (0.9 pts.) Ompliu la taula següent indicant, per a cada vèrtex del DAG donat, el seu nivell. Quant val la profunditat del DAG? No cal justificar res.



|          |   |   |   |   |   |   |   |   |               |                      |
|----------|---|---|---|---|---|---|---|---|---------------|----------------------|
| nivell : | A | B | C | D | E | F | G | H | profunditat : | <input type="text"/> |
|          |   |   |   |   |   |   |   |   |               |                      |

- (b) (0.4 pts.) Per a cada afirmació donada a continuació, marqueu amb una X la casella corresponent segons si és certa o falsa. No cal justificar res.

*Nota:* Cada resposta correcta sumarà 0.1 punts; cada resposta equivocada restarà 0.1 punts, llevat del cas que hi hagi més respostes equivocades que correctes, en què la nota de l'exercici serà 0.

- (1) Per a tot vèrtex  $u$  d'un DAG  $G$ , si  $u$  és una fulla (vèrtex sense successors) llavors  $\text{nivell}(u) = \text{profunditat}(G)$ .
- (2) Per a tot vèrtex  $u$  d'un DAG  $G$ , si  $\text{nivell}(u) = \text{profunditat}(G)$  llavors  $u$  és una fulla.
- (3) La profunditat d'un DAG amb  $n$  vèrtexs és  $O(n)$ .
- (4) La profunditat d'un DAG amb  $n$  vèrtexs és  $\Omega(\log n)$ .

|      | (1) | (2) | (3) | (4) |
|------|-----|-----|-----|-----|
| CERT |     |     |     |     |
| FALS |     |     |     |     |

- (c) (1.7 pts.) En aquest exercici assumirem que els grafs es representen amb llistes d'adjacències, i que el vèrtexs s'identifiquen amb naturals consecutius 0, 1, etc.

Ompliu els buits de la funció següent:

```
vector<int> levels(const vector<vector<int>>& G);
```

que, donat un DAG  $G = (V, E)$ , retorna un vector que, per a cada vèrtex  $u \in V$ , conté el valor  $\text{nivell}(u)$  a la posició  $u$ . Doneu i justifiqueu el cost temporal en el cas pitjor en termes de  $n = |V|$  i  $m = |E|$ .

```

vector<int> levels(const vector<vector<int>>& G) {
 int n = G.size ();
 vector<int> lvl(n, -1), pred(n, 0);

 for (int u = 0; u < n; ++u)
 for (int v : G[u])
 ++pred[v];

 queue<int> Q;
 for (int u = 0; u < n; ++u)
 if (pred[u] == 0) {
 Q.push(u);

 }

 while (not Q.empty()) {
 int u = Q.front (); Q.pop();
 for (int v : G[u]) {
 --pred[v];

 if (pred[v] == 0) Q.push(v);
 }
 }
 return lvl ;
}

```

Cost i justificació:

#### Problema 4

(3 pts.)

Donat  $n > 0$ , un *desarranjament* (de mida  $n$ ) és una permutació de  $\{0, \dots, n-1\}$  sense punts fixos; és a dir,  $\pi$  és un desarranjament si i només si  $\pi(i) \neq i$  per tot  $i, 0 \leq i < n$ . Per exemple  $\pi = (2, 0, 3, 1)$  és un desarranjament, però  $\pi' = (1, 3, 2, 0)$  no (ja que  $\pi'(2) = 2$ ).

Completeu el codi C++ següent per a generar tots els desarranjaments de mida  $n$ . No cal justificar res.

```

void write(const vector<int>& p, int n) {
 for (int k = 0; k < n; ++k) cout << " " << p[k];
 cout << endl;
}

void generate(int k, int n, vector<int>& p, vector<bool>& used) {
 if () write(p, n);
 else {
 for (int i = 0; i < n; ++i)
 if () {

 generate(k+1, n, p, used);

 }
 }
};

void generate_all (int n) {
 vector<int> p(n);
 vector<bool> used(n, false);
 generate(, n, p, used);
}

```

Examen Final EDA

Duració: 3 hores

19/01/2018

## Problema 1

(2.5 pts.)

En aquest exercici, per *heaps* entendrem *min-heaps* de nombres enters. També assumirem la seva implementació habitual, en què un arbre binari complet amb  $n$  elements es representa amb un vector de mida  $n + 1$  la primera component del qual (la d'índex 0) no s'usa.

- (a) (1.25 pts.) Sigui  $v$  un vector de mida  $n + 1$  que representa un arbre binari complet amb  $n$  enters. A més, excepte possiblement a un cert node amb índex  $i$  (on  $1 \leq i \leq n$ ), compleix la propietat de *min-heap*: si un node té valor  $x$  i un descendent seu té valor  $y$ , llavors  $x \leq y$ .

Implementeu en C++ la funció

```
void shift_down(vector<int>& v, int i);
```

que enfonsa el node amb índex  $i$  fins a restaurar la propietat de *min-heap* a tots els nodes.

- (b) (1.25 pts.) Implementeu en C++ una funció

```
void update (vector<int>& v, int i, int x);
```

que donats:

- un vector  $v$  que representa un heap amb  $n$  enters,
- un índex  $i$  tal que  $1 \leq i \leq n$ , i
- un enter  $x$ ,



assigna al node amb índex  $i$  el valor  $x$  (descartant el valor anterior en el node) i actualitza  $v$  per tal que continuï representant un heap.

Podeu utilitzar la funció *shift\_down* de l'apartat anterior, així com també la següent funció *shift\_up*:

```
void shift_up (vector<int>& v, int i) {
 if (i != 1 and v[i/2] > v[i]) {
 swap(v[i], v[i/2]);
 shift_up(v, i/2);
 }
}
```

## Problema 2

(2 pts.)

Donada una permutació  $\pi$  de  $\{1, \dots, n\}$ , el valor  $\pi(i)$  és un *màxim esquerre* si  $\pi(i) > \pi(j)$  per tot  $j$ ,  $1 \leq j < i$ . El primer element  $\pi(1)$  és sempre un màxim esquerre.

Per exemple  $\pi = (3, 1, 5, 4, 6, 2)$  té 3 màxims esquerres (concretament  $\pi(1) = 3$ ,  $\pi(3) = 5$  i  $\pi(5) = 6$ ).

Completeu el següent codi C++ que escriu les permutacions de  $\{1, \dots, n\}$  que tenen exactament  $k$  màxims esquerres, on  $1 \leq k \leq n$ .

```
int n, k;
vector<int> perm;
vector<bool> used;

void write() {
 cout << perm[1];
```

```

 for (int i = 2; i ≤ n; ++i) cout << ' ' << perm[i];
 cout << endl;
}

void generate(int i, int lm, int mx) {
 if ([] > k or [] < k) return;
 if ([]) write();
 else {
 for (int j = 1; j ≤ n; ++j)
 if ([]) {
 []
 perm[i] = [];
 if ([]) generate(i+1, lm+1, j);
 else generate(i+1, lm, mx);
 []
 }
 }
}

int main() {
 cin >> n >> k;
 perm = vector<int>(n+1);
 used = vector<bool>(n+1, false);
 generate(1, 0, 0);
}

```

### Problema 3

(3 pts.)

(a) (1 pt.) Considereu el següent programa:

```

int x = ...; // s'inicialitza amb algun valor
int mystery(const string& s) {
 int h = 0;
 for (int k = s.size() - 1; k ≥ 0; --k)
 h = x*h + int(s[k]);
 return h;
}

```

Què calcula la funció *mystery*?

Si es volgués fer servir la funció *mystery* com a funció de hash per al tipus **string**, seria 0 un valor adequat per a  $x$ ? Per què?

- (b) (1 pt.) Per a cada afirmació donada a continuació, marqueu amb una X la casella corresponent segons si és certa o falsa. No cal justificar res.

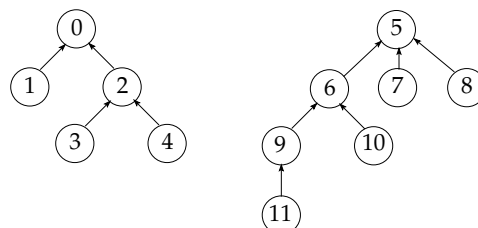
*Nota:* Cada resposta correcta sumará 0.25 punts; cada resposta equivocada restará 0.25 punts, llevat del cas que hi hagi més respostes equivocades que correctes, en què la nota de l'exercici serà 0.

Recordeu: SAT és el problema de, donada una fórmula proposicional, dir si és satisfactible o no; i HAM és el problema de, donat un graf, dir si és hamiltonià.

- (1) Si existeix un problema NP que pertany a P aleshores  $P = NP$ .
- (2) Si hi ha una reducció polinòmica d'un problema X a SAT llavors hi ha una reducció polinòmica de X a HAM.
- (3) Hi ha problemes de la classe NP per als quals no es coneix cap algorisme que els resolgui.
- (4) Si un problema X es pot reduir polinòmicament a un problema  $Y \in P$  aleshores  $X \in P$ .

|      | (1) | (2) | (3) | (4) |
|------|-----|-----|-----|-----|
| CERT |     |     |     |     |
| FALS |     |     |     |     |

- (c) (0.5 pts.) Considereu el següent bosc d'arbres arrelats, que representa a una col·lecció de conjunts disjunts:



El valor retornat en cridar *find*(2) és .

El valor retornat en cridar *find*(9) és .

- (d) (0.5 pts.) Considereu la següent variant de la cerca dicotòmica que, donats:

- un element  $x$ ,

- un vector  $v$  ordenat de forma no-decreixent, i
- dues posicions  $l$  i  $r$ ,

retorna una posició entre  $l$  i  $r$  en què  $x$  apareix a  $v$  (o  $-1$  si no n'hi ha cap):

```
int posicio (double x, const vector<double>& v, int l, int r) {
 if (l > r) return -1;
 int p = l + int((r-l)/3.0);
 if (x < v[p]) return posicio (x, v, l, p-1);
 if (x > v[p]) return posicio (x, v, p+1, r);
 return p;
}
```

Analitzeu el cost asimptòtic en el cas pitjor.

#### Problema 4

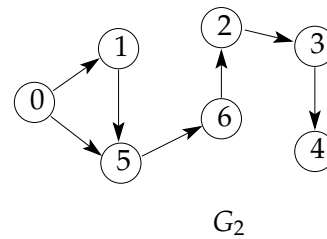
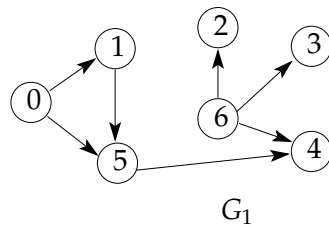
(2.5 pts.)

Donats dos digrafs  $G_1 = (V, E_1)$  i  $G_2 = (V, E_2)$  definits sobre el mateix conjunt de vèrtexs  $V$ , anomenem la *composició* de  $G_1$  amb  $G_2$  al digraf que representarem com  $G_1 \circ G_2$  i que té com a vèrtexs  $V$ , i com a arestes el següent conjunt:

$$E = \{(u, w) \mid \exists v \text{ tal que } (u, v) \in E_1 \wedge (v, w) \in E_2\}$$

*Nota.* En aquest exercici es permet que els digrafs tinguin *auto-cicles*, és a dir, arestes de la forma  $(u, u)$ .

(a) (0.5 pts.) Considereu els següents digrafs  $G_1$  i  $G_2$ :



Dibuixeu  $G_1 \circ G_2$ .

(b) (1 pt.) Suposem que  $G_1$  i  $G_2$  estan representats amb matrius d'adjacència. Implementeu en C++ una funció

```

typedef vector<vector<int>> matrix;
matrix comp(const matrix& G1, const matrix& G2);

```

que, donades les matrius d'adjacència de  $G_1$  i de  $G_2$ , calculi la matriu d'adjacència de  $G_1 \circ G_2$  en temps  $\Theta(|V|^3)$  en el cas pitjor. Justifiqueu que el cost és el demanat.



(c) (1 pt.) Expliqueu com implementar la funció *comp* en temps millor que  $\Theta(|V|^3)$ .



Examen Final EDA

Duració: 3 hores

20/06/2018

## Problema 1

(2 pts.)

(a) (0.5 pts.) La classe *stack*  $\langle T \rangle$  de la STL té dues funcions membre *top*:

```
T& top();
const T& top() const;
```

En canvi, la classe *priority\_queue*  $\langle T \rangle$  només té una funció *top*:

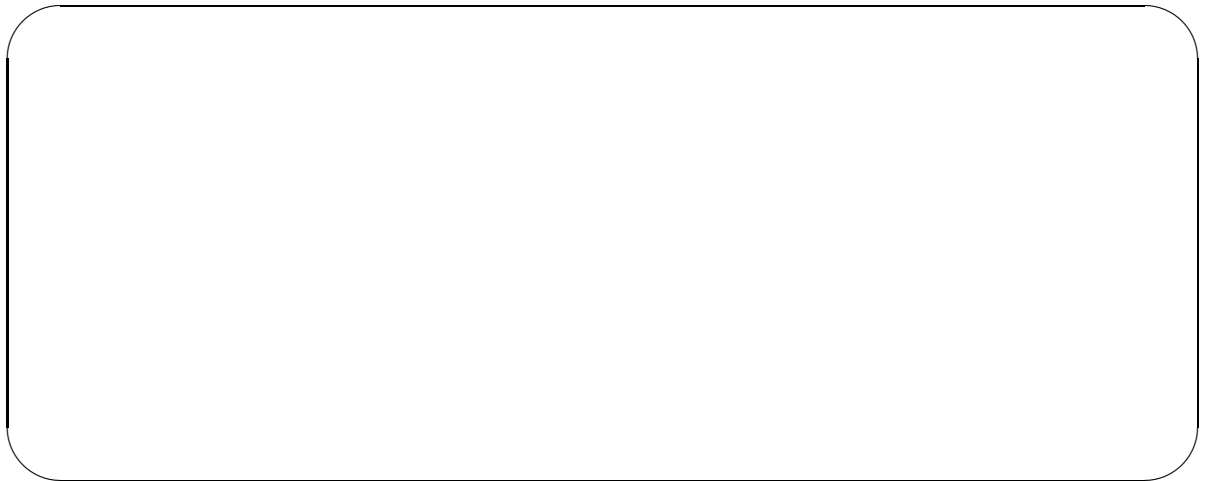
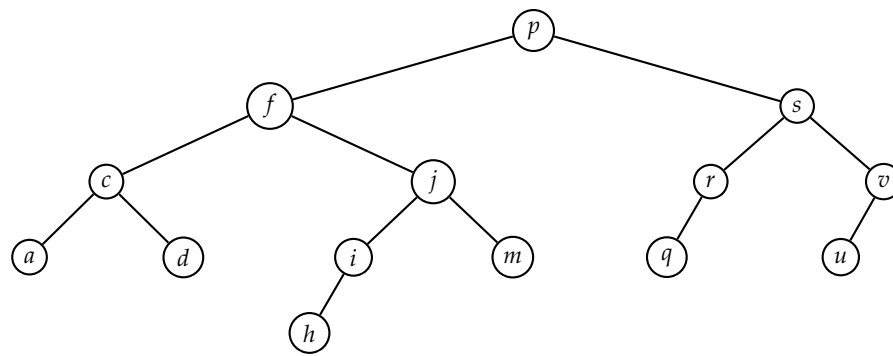
```
const T& top() const;
```

Quin inconvenient hi ha en què la classe *priority\_queue*  $\langle T \rangle$  tingui una funció *T& top()*?

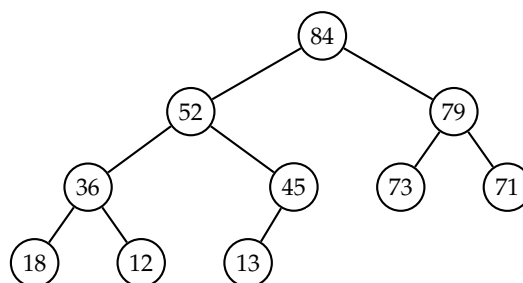
(b) (0.5 pts.) Per a l'alfabet  $\{A, B, C, D\}$  i la taula de freqüències absolutes següent per a 100 caràcters, doneu un codi de Huffman i dibuixeu l'arbre corresponent. No cal justificar res.

|   |    |
|---|----|
| A | 70 |
| B | 10 |
| C | 15 |
| D | 5  |

(c) (0.5 pts.) Dibueixeu l'arbre AVL resultant d'inserir les tres claus  $x, y, z$  en aquest ordre a l'arbre AVL de la figura. Noteu que les claus s'ordenen per ordre alfabètic. No cal justificar res.



(d) (0.5 pts.) Donat el max-heap següent:



dibuixeu el max-heap resultant d'afegir 55 i d'esborrar l'element màxim (en aquest ordre). No cal justificar res.



**Problema 2****(3 punts)**

- (a) (1 pt.) Considereu una recurrència de la forma

$$T(n) = T(n/b) + \Theta(\log^k n)$$

on  $b > 1$  i  $k \geq 0$ . Demostreu que la seva solució és  $T(n) = \Theta(\log^{k+1} n)$ .

Justificació:

*Nota:*  $\log^k n$  és una forma breu d'escriure  $(\log(n))^k$ .

*Pista:* feu un canvi de variable.

- (b) (1 pt.) Donat un  $n \geq 1$ , es diu que una seqüència de  $n$  enters  $a_0, \dots, a_{n-1}$  és *unimodal* si existeix  $t$  amb  $0 \leq t < n$  tal que  $a_0 < \dots < a_{t-1} < a_t$  i  $a_t > a_{t+1} > \dots > a_{n-1}$ . A l'element  $a_t$  se l'anomena el *cim* de la seqüència. Per exemple, la seqüència 1,3,5,9,4,1 és unimodal, i el seu cim és 9 (preneu  $t = 3$ ).

Ompliu els buits del codi a continuació per tal que la funció

```
bool search(const vector<int>& a, int x),
```

donat un vector no buit  $a$  que conté una seqüència unimodal i un enter  $x$ , retorni si  $x$  apareix a la seqüència o no. No cal justificació.

```
bool search(const vector<int>& a, int x, int l, int r) {
 if () return x == a[l];
 int m = (l+r)/2;
 auto beg = a.begin ();
 if (a[m] <)
 return search(a, x, m+1, r) or binary_search(beg + l, beg + , x);
 else
 return search(a, x, l, m) or binary_search(beg + , beg + r + 1, x);
}

bool search(const vector<int>& a, int x) { return search(a, x, 0,); }
```

*Nota:* Donats un element  $x$  i iteradors  $first, last$  tals que l'interval  $[first, last)$  està ordenat (creixentment o decreixentment), la funció  $binary\_search(first, last, x)$  retorna **true** si  $x$  apareix a l'interval  $[first, last)$  i **false** altrament, en temps logarítmic en la mida de l'interval en el cas pitjor.

- (c) (1 pt.) Analitzeu el cost en temps en el cas pitjor d'una crida  $search(a, x)$ , on  $a$  és un vector de mida  $n > 0$  que conté una seqüència unimodal i  $x$  és un enter. Descriviu una situació en què es pugui donar aquest cas pitjor.

**Problema 3****(3 pts.)**

Un estudiant d'EDA s'està preparant per a l'examen de laboratori i s'encalla en un problema de la llista "Graph Algorithms" del Jutge. Després de rumiar-hi una mica, veu que el que cal és, essencialment, donat un graf (no dirigit, representat amb llistes d'adjacència), determinar si és 2-colorable; és a dir, si és possible assignar un color *red* o *blue* a cada vèrtex del graf, de manera que vèrtexs adjacents no estiguin pintats amb el mateix color.

Malauradament, el Jutge respon Time Limit Exceeded a tots els enviaments que fa. Arribat a aquest punt, l'estudiant decideix buscar per Internet algun codi més eficient, i troba el següent penjat en un fòrum:

```
bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
 vector<bool>& marked, bool is_red) {
 col[u] = is_red ;
 marked[u] = true;
 for (int v : g[u]) {
 if (not marked[v]) {
 if (not two_col_aux(g, v, col, marked, not is_red)) return false;
 }
 }
 return true;
```

```

}

bool two_colourable (const vector<vector<int>>& g) {
 int n = g.size ();
 vector<bool> marked(n, false);
 vector<bool> col(n);
 for (int u = 0; u < n; ++u) {
 if (not marked[u]) {
 if (not two_col_aux(g, u, col, marked, false)) return false;
 }
 }
 return true;
}

```

Ara s'espera que aquest codi serà prou eficient, perquè després d'una mica d'anàlisi veu que triga  $O(|V| + |E|)$ . L'envia al Jutge, i resulta que... Wrong Answer!

- (a) (2 pts.) Escriviu una versió correcta de *two\_col\_aux* de manera que *two\_colorable*, ara sí, determini si un graf és 2-colorable en temps  $O(|V| + |E|)$ .

```

bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
 vector<bool>& marked, bool is_red)

```

- (b) (1 pt.) Finalment l'estudiant troba com arreglar el codi i que el Jutge l'hi accepti. Mos-

quejat, continua llegint el fòrum. L'autor del codi erroni afirma que el seu programa es pot estendre fàcilment al problema de 3-colorabilitat amb el mateix cost asimptòtic. Us ho creieu? Per què?

**Problema 4****(2 pts.)**

Considereu el següent problema: donats  $n$  nombres enters positius  $a_0, a_1, \dots, a_{n-1}$  i dos altres enters  $l$  i  $u$  tals que  $0 \leq l \leq u \leq \sum_{i=0}^{n-1} a_i$ , es tracta de trobar totes les possibles solucions  $(x_0, x_1, \dots, x_{n-1}) \in \{0, 1\}^n$  de la doble inequació:

$$l \leq a_0x_0 + a_1x_1 + \dots + a_{n-1}x_{n-1} \leq u$$

(a) (1 pt.) Completeu el programa següent perquè resolgui el problema anterior:

```

int n, l, u;
vector<int> a;
vector<bool> x;

void sols (int k, int sum_chosen, int sum_rest) {
 if (k == n) {
 for (int i = 0; i < n; ++i) cout << x[i];
 cout << endl;
 } else {
 if () {
 x[k] = 0; sols (k+1, sum_chosen, sum_rest - a[k]);
 }
 if () {
 x[k] = 1; sols (k+1, sum_chosen + a[k], sum_rest - a[k]);
 }
 }
}

int main() {
 cin >> n >> l >> u;
 a = vector<int>(n);
 for (int i = 0; i < n; ++i) cin >> a[i];

 int s = 0;
 for (int i = 0; i < n; ++i) s += a[i];

 x = vector<bool>(n);
 sols (, ,);
}

```

- (b) (1 pt.) El programa de l'apartat anterior escriu les solucions en ordre lexicogràfic **creixent**. És a dir, per a l'entrada  $n = 3$ ,  $l = 3$ ,  $u = 5$  i  $a_i = i + 1$  ( $0 \leq i \leq 2$ ), es produeix la sortida:

001  
011  
101  
110

Expliqueu com modificar la funció *sols* perquè s'escriuin les solucions en ordre lexicogràfic **decreixent**, és a dir en l'ordre invers de l'anterior, sense usar espai addicional. En particular, no es considerarà com a resposta vàlida l'algorisme consistent a guardar tota la sortida en un vector, i al final ordenar-lo i escriure'l.

Examen Final EDA

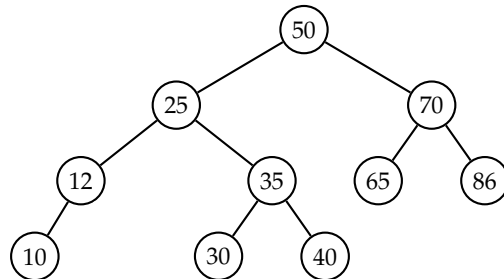
Duració: 3 hores

14/01/2019

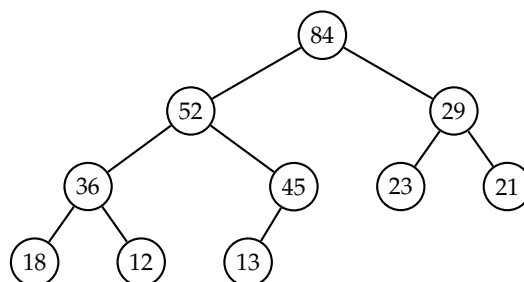
## Problema 1

(3.5 pts.)

(a) (0.75 pts.) Dibuixeu l'arbre AVL resultant d'afegir 31 a l'arbre AVL següent:



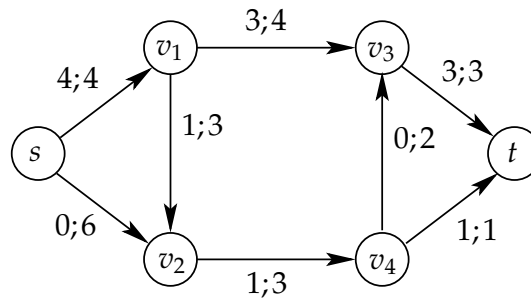
(b) (0.75 pts.) Dibuixeu el heap resultant d'eliminar el màxim del max-heap següent:

(c) (0.5 pts.) La solució de la recurrència  $T(n) = 3T(n/9) + \Theta(n)$  és

(d) (0.5 pts.) La solució de la recurrència  $T(n) = 3T(n/9) + \Theta(\sqrt{n})$  és

(e) (0.5 pts.) La solució de la recurrència  $T(n) = 3T(n/9) + \Theta(1)$  és

(f) (0.5 pts.) Considereu la xarxa de flux següent amb origen  $s$  i destí  $t$ , i el flux indicat:



Per a cada arc, s'indiquen el flux al llarg de l'arc i la seva capacitat, separats per punt i coma. Així, l'arc de  $v_1$  a  $v_3$  porta 3 unitats de flux i té capacitat 4.

Dibuixeu la xarxa residual d'aquesta xarxa respecte al flux donat.

## Problema 2

(4 pts.)

En aquest exercici treballarem amb un tipus *polynomial* per representar polinomis en una variable  $x$  amb coeficients enters.

(a) (0.5 pts.) Suposem que implementem els polinomis amb vectors d'enters (o sigui, *polynomial* és **vector<int>**), de forma que per a  $k = 0, 1, \dots$  el  $k$ -èsim valor del vector és el coeficient de  $x^k$ . Així, per exemple el vector  $\{1, 0, -3\}$  representaria el polinomi  $1 - 3x^2$ .



Donats un polinomi  $p$  i enters  $c \neq 0$  i  $k \geq 0$ , considereu la funció següent:

```
polynomial mystery(const polynomial& p, int c, int k) {
 int n = p.size ();
 polynomial q(n + k, 0);
 for (int i = 0; i < n; ++i)
 q[i + k] = c*p[i];
 return q;
}
```

Responen:

la funció *mystery* calcula  i el seu cost en temps en termes de  $n = p.size()$  i  $k$  és  $\Theta(\text{  })$ .

- (b) (1.5 pts.) Suposem que ara implementem els polinomis amb un diccionari amb claus enteres i valors enteres, de forma que un parell del diccionari amb clau  $k$  i valor  $v$  (amb  $v \neq 0$ ) representa el monomi  $v \cdot x^k$ . Així, per exemple el diccionari amb parells clau-valor  $\{(0,1), (2,-3)\}$  representaria el polinomi  $1 - 3x^2$ .

Més concretament, assumim que *polynomial* és *unordered\_map<int,int>*.

Quina de les estructures de dades vistes a classe es podria usar per implementar els *unordered\_map<int,int>*?

Ompliu els buits a la reimplementació següent de la funció *mystery* que usa aquesta nova representació dels polinomis.

```
polynomial mystery(const polynomial& p, int c, int k) {
 polynomial q;
 for (auto mon : p)
 q[] = ;
 return q;
}
```

Doneu i justifiqueu una fita superior acurada del cost en temps en el cas mig.

- (c) (0.25 pts.) Des d'aquest apartat en endavant suposem de nou que implementem els polinomis amb vectors d'enters (és a dir, *polynomial* és **vector<int>**).

Considereu la funció següent, que defineix l'operador  $+$  per a polinomis per poder escriure en C++ la suma de dos polinomis  $p$  i  $q$  com  $p + q$ :

```
polynomial operator+(const polynomial& p, const polynomial& q) {
 polynomial s(max(p.size(), q.size()));
 for (int i = 0; i < p.size(); ++i)
 s[i] += p[i];
 for (int i = 0; i < q.size(); ++i)
 s[i] += q[i];
 return s;
}
```

Sigui  $n = \max(p.size(), q.size())$ .

Responen:

el cost de la funció en termes de  $n$  és  $\Theta(\text{ } \text{ } \text{ } )$ .

- (d) (1.75 pts.) Ompliu els buits de la funció següent, que defineix l'operador  $*$  per a polinomis per poder escriure en C++ el producte de dos polinomis  $p$  i  $q$  com  $p * q$ . Podeu usar les funcions que apareixen en els apartats anteriors. Suposeu que  $p.size() = q.size() = n$ , on  $n \geq 1$  és una potència de 2. Cal que el cost de la funció sigui **estrictament millor** que  $\Theta(n^2)$ .

```
polynomial operator*(const polynomial& p, const polynomial& q) {
 int n = p.size();
 if (n == 1) return ;
 int n2 = n/2;

 polynomial p0(n2), p1(n2);
 for (int k = 0; k < n2; ++k) p0[k] = p[k];
 for (int k = n2; k < n; ++k) p1[k - n2] = p[k];

 polynomial q0(n2), q1(n2);
 for (int k = 0; k < n2; ++k) q0[k] = q[k];
 for (int k = n2; k < n; ++k) q1[k - n2] = q[k];

 polynomial p0_q0 = p0 * q0;
 polynomial p1_q1 = p1 * q1;
```

}

Analitzeu el cost de la funció en termes de  $n$ .

### Problema 3

(2.5 pts.)

Donat un natural  $N \geq 1$ , considerem el conjunt dels  $N$  primers nombres naturals  $\{0, \dots, N - 1\}$ . Un *recobriment per conjunts* (en anglès, *set cover*) de  $\{0, \dots, N - 1\}$  és una col·lecció de subconjunts  $S_0, \dots, S_{M-1}$  de  $\{0, \dots, N - 1\}$  tal que cadascun dels  $N$  primers nombres naturals està inclòs en algun dels subconjunts, és a dir,  $\bigcup_{i=0}^{M-1} S_i = \{0, \dots, N - 1\}$ .

El problema de SET-COVER consisteix a, donats:

- dos naturals  $K$  i  $N$ ,
- subconjunts  $S_0, \dots, S_{M-1}$  de  $\{0, \dots, N - 1\}$  que recobreixen (per conjunts)  $\{0, \dots, N - 1\}$ ,

determinar si, dels subconjunts donats, es poden escollir  $k$  subconjunts  $S_{i_0}, S_{i_1}, \dots, S_{i_{k-1}}$  amb  $k \leq K$  que també formin un recobriment (per conjunts) de  $\{0, \dots, N - 1\}$ .

Per exemple, suposem que  $N = 4$ ,  $S_1 = \{0, 1, 3\}$ ,  $S_2 = \{0, 3\}$  i  $S_3 = \{0, 2, 3\}$ . Llavors si  $K = 2$  la resposta és TRUE (perquè  $S_1 \cup S_3 = \{0, 1, 2, 3\}$ ), però si  $K = 1$  la resposta és FALSE (perquè no hi ha cap  $S_i$  que tot sol cobreixi el conjunt  $\{0, 1, 2, 3\}$ ).

(a) (0.75 pts.) Considereu la implementació següent de la funció

```
bool set_cover (int K, int N, const vector<set<int>>& S);
```

per resoldre el problema de SET-COVER:

```
bool set_cover_rec (int K, int N, const vector<set<int>>& S,
 vector<int>& w, int c) {
 if (c == N) return true;
 if (K == 0) return false;
 for (int i = 0; i < S.size (); ++i) {
 for (int x : S[i]) {
 if (w[x] == 0) ++c;
 ++w[x];
 }
 if (set_cover_rec (K-1, N, S, w, c)) return true;
 for (int x : S[i]) {
 --w[x];
 if (w[x] == 0) --c;
 } }
 return false;
}

bool set_cover (int K, int N, const vector<set<int>>& S) {
 vector<int> w(N, 0);
 return set_cover_rec (K, N, S, w, 0);
}
```

Aquesta implementació té un greu error d'eficiència. Expliqueu quin és aquest error i reescriuiu el codi anterior fent els canvis necessaris per corregir-lo.

- (b) (1 pt.) Donat un graf  $G = (V, E)$ , un *recobriment per vèrtexs* (en anglès, *vertex cover*) és un subconjunt de vèrtexs  $W \subseteq V$  tal que tota aresta  $\{u, v\} \in E$  té almenys algun dels dos extrems a  $W$ , és a dir,  $u \in W$  o  $v \in W$ .

El problema de VERTEX-COVER consisteix a, donats un natural  $K$  i un graf  $G = (V, E)$ , determinar si existeix un subconjunt  $W \subseteq V$  de com a molt  $K$  vèrtexs que sigui un recobriments (per vèrtexs) de  $G$ .

De les dues funcions definides a continuació, una constitueix una reducció de VERTEX-COVER a SET-COVER i l'altra no. Demostreu que la que és correcta ho és i doneu un contraexemple per a la incorrecta.

(1) Donat  $G = (V, E)$ , amb  $V = \{u_0, \dots, u_{N-1}\}$ ,

$$f(K, G) = (K, N, S),$$

on  $S = \{S_e \mid e \in E\}$  i  $S_e = \{i \mid u_i \in e\}$ .

(2) Donat  $G = (V, E)$ , amb  $E = \{e_0, \dots, e_{N-1}\}$ ,

$$g(K, G) = (K, N, S),$$

on  $S = \{S_u \mid u \in V\}$  i  $S_u = \{i \mid u \in e_i\}$ .

Reducció correcta i demostració:

Reducció incorrecta i contraexemple:

- (c) (0.75 pts.) Supposeu que hem pogut reduir VERTEX-COVER a SET-COVER, i que sabem que VERTEX-COVER és NP-complet. Podem deduir només a partir d'això que SET-COVER és un problema NP-complet? En cas afirmatiu, justifiqueu-ho. Altrament demostreu que SET-COVER és NP-complet (assumint que VERTEX-COVER es redueix a SET-COVER i que VERTEX-COVER és NP-complet).

## Examen Final EDA

Duració: 3 hores

17/06/2019

## Problema 1

(3 pts.)

(a) (1.25 pts.) Denotem amb  $\wedge$  la AND lògica, amb  $\vee$  la OR lògica i amb  $\neg$  la NOT lògica. Donades variables booleanes  $x_1, \dots, x_n$ , definim la fórmula  $\text{xor}(x_1, \dots, x_n)$  recursivament així:

- Si  $n = 1$  llavors  $\text{xor}(x_1) = x_1$ ;
- Altrament:

$$\text{xor}(x_1, \dots, x_n) = \left( (\text{xor}(x_1, \dots, x_{\lceil \frac{n}{2} \rceil}) \wedge \neg \text{xor}(x_{\lceil \frac{n}{2} \rceil+1}, \dots, x_n)) \vee (\neg \text{xor}(x_1, \dots, x_{\lceil \frac{n}{2} \rceil}) \wedge \text{xor}(x_{\lceil \frac{n}{2} \rceil+1}, \dots, x_n)) \right).$$

Per exemple,  $\text{xor}(x_1, x_2) = ((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2))$ , i

$$\text{xor}(x_1, x_2, x_3) = \left( \left( ((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)) \wedge \neg x_3 \right) \vee \left( \neg((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)) \wedge x_3 \right) \right)$$

Doneu raonadament una expressió en notació asimptòtica  $\Theta$  de la mida de la fórmula  $\text{xor}(x_1, \dots, x_n)$  en funció de  $n$ . Assumiu que cada variable booleana, així com cada símbol (parèntesis,  $\wedge$ ,  $\vee$ ,  $\neg$ ) ocupa 1 byte.

(b) (1.25 pts.) Fixat un natural  $k \geq 2$ , el problema de  $k$ -COLOR consisteix en, donat un graf no dirigit  $G = (V, E)$ , determinar si existeix un  $k$ -colorejat del graf; això és, una funció  $C : V \rightarrow \{1, \dots, k\}$  tal que per tot  $\{u, v\} \in E$  es compleix  $C(u) \neq C(v)$ .

Volem justificar que  $k$ -COLOR es redueix a  $k + 1$ -COLOR. Considereu la proposta de reducció següent:



"Donat un graf  $G$ , definim la seva imatge per la reducció com el mateix graf. És clar que és un algorisme que triga temps polinòmic en la mida de l'entrada. I si  $G$  admet un  $k$ -colorejat, aleshores aquesta mateixa assignació de colors és un  $(k + 1)$ -colorejat."

És correcta aquesta reducció? Si no ho és, per què?

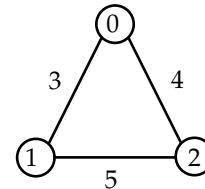
- (c) (0.5 pts.) A la base de dades d'una agència matrimonial hi ha un conjunt  $H$  d'homes i un conjunt  $D$  de dones. L'agència també disposa de la llista  $L$  de parells  $(h, d)$ , amb  $h \in H$  i  $d \in D$ , tals que  $h$  i  $d$  estarien disposats a casar-se entre si. Cada  $h$  i cada  $d$  es pot casar com a molt una vegada. Considerem el problema de trobar el nombre més gran de matrimonis que es poden formar. Indiqueu a quin problema conegut de teoria de grafs es tradueix aquest problema.

**Problema 2****(3 pts.)**

En aquest problema considerem grafs no dirigits amb costos enters positius a les arestes. Els vèrtexs s'identifiquen amb nombres consecutius 0, 1, 2, etc. Representem els grafs amb llistes d'adjacència usant vectors de vectors de parells d'ints, on el primer component és el cost de l'aresta i el segon, el vèrtex adjacent:

```
typedef pair<int,int> CostVertex;
typedef vector<vector<CostVertex>> Graph;
```

(a) (0.75 pts.) Considereu el graf mostrat a continuació:



Completeu el codi següent perquè la variable *g* prengui per valor el graf donat.

*Graph* *g* = {  };

(b) (0.75 pts.) Considereu el fragment de codi següent:

```
vector<CostVertex> mystery(const Graph& g) {
 int n = g.size();
 vector<CostVertex> t(n, {-1, -1});
 vector<int> a(n, INT_MAX);
 a[0] = 0;
 vector<bool> marked(n, false);
 priority_queue<CostVertex, vector<CostVertex>, greater<CostVertex>> pq;
 pq.push({0, 0});
 while (not pq.empty()) {
 int u = pq.top().second;
 pq.pop();
 if (not marked[u]) {
 marked[u] = true;
 for (CostVertex cv : g[u]) {
 int c = cv.first;
 int v = cv.second;
 if (a[v] > a[u] + c) {
 a[v] = a[u] + c;
 t[v] = {c, u};
 pq.push({a[v], v});
 }
 }
 }
 }
 return t;
}
```

Al final de l'execució de la funció *mystery*, quin significat té el valor de *a[u]* per a un vèrtex *u* donat? No cal justificació.

(c) (1.5 pts.) Considerem el codi de l'apartat (b). Suposem que  $g$  és connex.

Si  $t$  és el vector retornat per  $mystery(g)$ , sigui  $T$  el conjunt d'arestes  $\{u, v, c\}$  tals que  $0 \leq u < n$ ,  $t[u] = \{c, v\}$  i  $v \neq -1$ . Es pot veure (no cal que ho demostreu) que  $T$  indueix un subgraf de  $g$  que és connex i sense cicles, o sigui un arbre.

Justifiqueu que  $T$  és un arbre **d'expansió**.

És necessàriament un arbre d'expansió **mínim**? Si la resposta és afirmativa, justifiqueu-ho. Si no, doneu-ne un contraexemple raonat.

**Problema 3****(2 pts.)**

Escriviu una funció *pred\_succ* que, donats un arbre binari de cerca *T* i un valor *x*, retorni dos apuntadors, un al node amb la clau més gran dels nodes amb clau menor o igual que *x*, i un altre apuntant al node amb la clau més petita dels nodes amb clau més gran o igual que *x*. Si no hi ha cap valor a *T* que sigui més petit o igual que *x*, cal que el primer apuntador retornat sigui *NULL*; similarment, si no hi ha cap valor a *T* que sigui més gran o igual que *x*, cal que el segon apuntador retornat sigui *NULL*.

Useu les definicions següents:

```
struct node {
 int key;
 node* left ;
 node* right ;
};
```

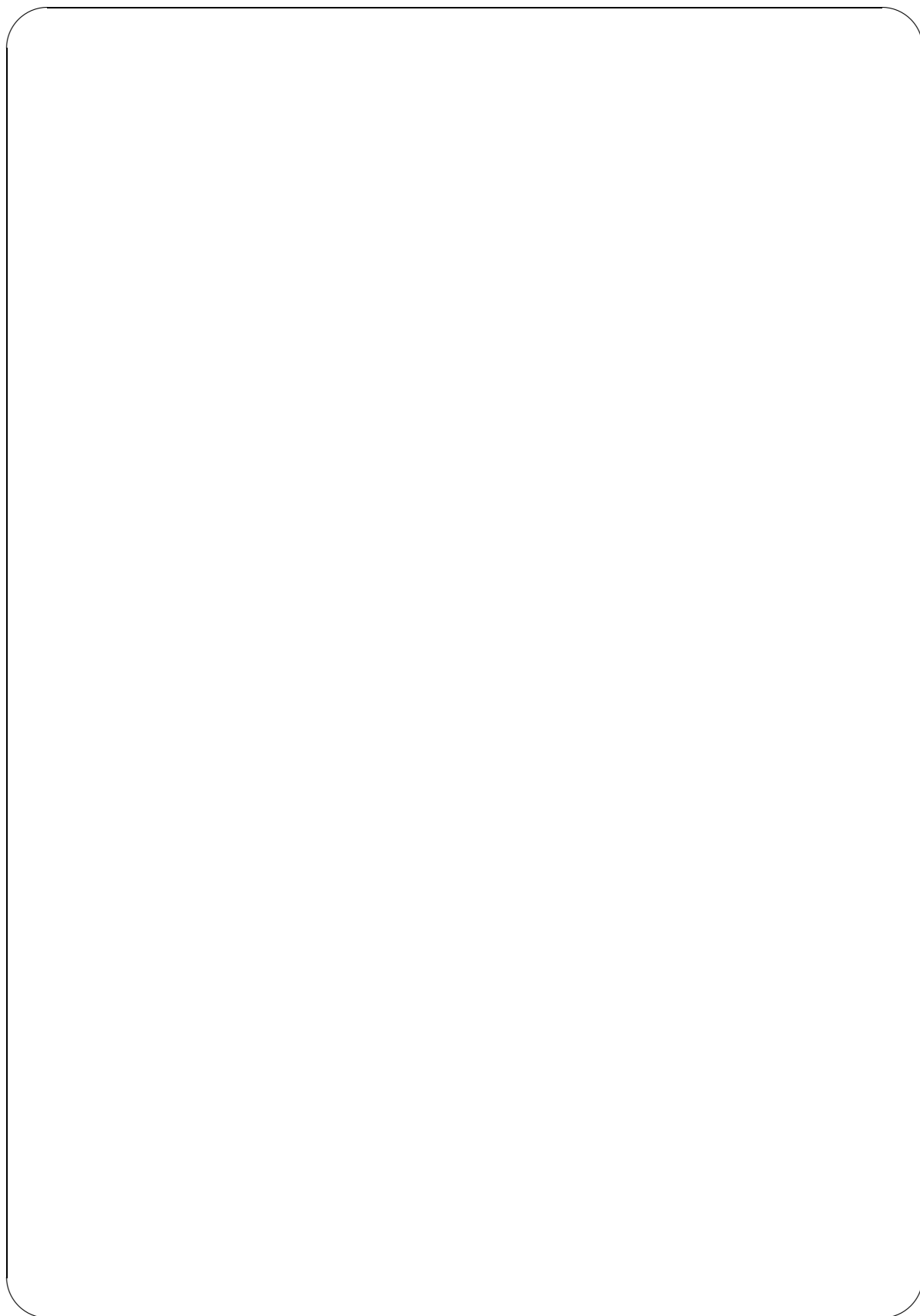
```
// Un arbre binari de cerca s'implementa amb un node*
```

```
pair <node*, node*> pred_succ(node* T, int x);
```

Per exemple, si *T* és un arbre binari de cerca buit, el resultat de *pred\_succ*(*T*,*x*) serà *<NULL, NULL>* independentment de *x*. Si *x* és a *T*, aleshores el resultat serà *<p, p>*, on *p* apunta al node que conté *x*. Si *x* no apareix a *T* i és estrictament més petit que qualsevol clau de *T*, i *T* és no buit, aleshores el resultat serà *<NULL, p>*, on *p* apunta al node amb la clau més petita de *T*, etc.

La vostra funció ha de tenir cost en temps  $O(h(T))$  per qualssevol *T* i *x*, on  $h(T)$  és l'alçada de *T*. No cal justificar el cost.

Si useu funcions auxiliars, implementeu-les també.



**Problema 4****(2 pts.)**

En el problema del viatjant, un viatjant ha de visitar els seus clients de  $n$  ciutats diferents. La distància entre la ciutat  $i$  i la ciutat  $j$  és un nombre positiu  $D[i][j]$ . El viatjant vol sortir de la seva pròpia ciutat, visitar una vegada i només una cadascuna de les altres ciutats, i finalment tornar al punt d'inici. El seu objectiu és fer això minimitzant la distància total del viatge.

- (a) (1.5 pts.) Completeu els buits en el programa donat a continuació perquè resolgui el problema del viatjant. Les ciutats s'identifiquen amb nombres consecutius 0, 1, 2, etc. Assumiu que la ciutat del viatjant és la ciutat 0.

```

struct TSP {

 vector<vector<double>> D;
 int n;
 vector<int> next, best_sol ;
 double best_tot_dist ;

 void recursive (int v, int t, double c) {
 if (t == n) {
 c += ;
 if (c < best_tot_dist) {
 best_tot_dist = ;
 best_sol = ;
 best_sol [v] = 0;
 }
 }
 else for (int u = 0; u < n; ++u)
 if (u ≠ v and next[u] == -1) {
 next[v] = u;
 recursive (u, t+1,);
 next[v] = ;
 }
 }

 TSP(const vector<vector<double>>& D) {
 this→D = D;
 n = D.size ();
 next = vector<int>(n, -1);
 best_tot_dist = DBL_MAX;
 recursive (0, 1, 0);
 }
};

```

(main a la pàgina següent)

```
int main () {
 int n;
 cin >> n;

 vector<vector<double>> D(n, vector<double>(n));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < n; ++j)
 cin >> D[i][j];

 TSP tsp(D);
 cout << "Best total distance: " << tsp.best_tot_dist << endl;
 vector<int> b = tsp.best_sol ;
 cout << "Sequence of cities: 0";
 int c = b[0];
 while (c != 0) {
 cout << ' ' << c;
 c = b[c];
 }
 cout << ' ' << 0 << endl;
}
```

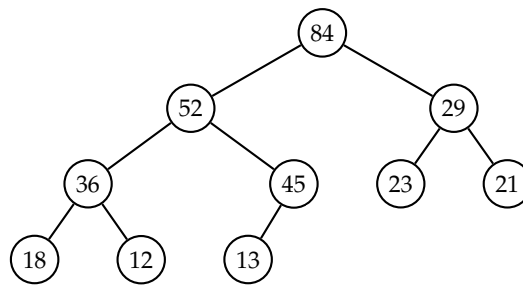
- (b) (0.5 pts.) Afegiu codi a la primera línia de la funció *recursive* per tal de fer més eficient el programa:

```
void recursive (int v, int t, double c) {
```

```
if (t == n) {
```

**Examen Final EDA**  
**Problema 1****Duració: 3 hores****13/01/2020**  
**(1.5 pts.)**

- (a) (0.5 pts.) Dibuixeu el heap resultant d'afegir l'element 64 al max-heap següent. No cal justificar la resposta.



- (b) (0.5 pts.) Quina és la **recurrència** que expressa el cost de l'algorisme d'Strassen per multiplicar matrius  $n \times n$ ? No cal justificar la resposta.

$T(n) =$

- (c) (0.5 pts.) Considereu el codi següent:

```
int g(int p);
int f(int n, int p) {
 if (n == 0) return p;
 else return 1 + f(n/2, g(p));
}
```

Si sabem que el cost de la funció  $g$  és quadràtic, quin és el cost asimptòtic en temps de  $f$  en funció de  $n$ ?



## Problema 2

(3 pts.)

Donat un graf dirigit acíclic (DAG)  $G$ , el *nivell* dels seus vèrtexs es defineix inductivament de la forma següent:

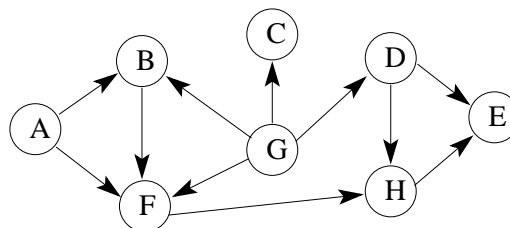
- si  $v$  és una arrel de  $G$  (un vèrtex sense predecessors) aleshores  $\text{nivell}(v) = 0$
- altrament,

$$\text{nivell}(v) = 1 + \max\{\text{nivell}(u) \mid u \text{ és un predecessor de } v\}$$

A més, la profunditat de  $G$  és el nivell més gran de qualsevol vèrtex:

$$\text{profunditat}(G) = \max\{\text{nivell}(v) \mid v \text{ vèrtex de } G\}$$

- (a) (0.5 pts.) Ompliu la taula següent indicant, per a cada vèrtex del DAG donat, el seu nivell. Quant val la profunditat del DAG? No cal justificar res.



nivell :

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |   |

profunditat :

- (b) (0.8 pts.) Per a cada afirmació donada a continuació, marqueu amb una X la casella corresponent segons si és certa o falsa. No cal justificar res.

*Nota:* Cada resposta correcta sumarà 0.2 punts; cada resposta equivocada restarà 0.2 punts, llevat del cas que hi hagi més respostes equivocades que correctes, en què la nota de l'exercici serà 0.

- (1) Per a tot vèrtex  $u$  d'un DAG  $G$ , si  $u$  és una fulla (vèrtex sense successors) llavors  $\text{nivell}(u) = \text{profunditat}(G)$ .
- (2) Per a tot vèrtex  $u$  d'un DAG  $G$ , si  $\text{nivell}(u) = \text{profunditat}(G)$  llavors  $u$  és una fulla.
- (3) La profunditat d'un DAG amb  $n$  vèrtexs és  $O(n)$ .
- (4) La profunditat d'un DAG amb  $n$  vèrtexs és  $\Omega(\log n)$ .

|      | (1) | (2) | (3) | (4) |
|------|-----|-----|-----|-----|
| CERT |     |     |     |     |
| FALS |     |     |     |     |

- (c) (1.7 pts.) En aquest problema assumirem que els grafs es representen amb llistes d'adjacències, i que el vèrtexs s'identifiquen amb naturals consecutius 0, 1, etc.

Ompliu els buits de la funció següent:

```
vector<int> levels(const vector<vector<int>>& G);
```

que, donat un DAG  $G = (V, E)$ , retorna un vector que, per a cada vèrtex  $u \in V$ , conté el valor  $\text{nivell}(u)$  a la posició  $u$ . Doneu i justifiqueu el cost temporal en el cas pitjor en termes de  $n = |V|$  i  $m = |E|$ .

```
vector<int> levels(const vector<vector<int>>& G) {
```

```
 int n = G.size ();
```

```
 vector<int> lvl(n, -1), pred(n, 0);
```

```
 for (int u = 0; u < n; ++u)
```

```
 for (int v : G[u])
```

```

```

```
 queue<int> Q;
```

```
 for (int u = 0; u < n; ++u)
```

```
 if (pred[u] == 0) {
```

```
 Q.push(u);
```

```

```

```
 }
```

```
 while (not Q.empty()) {
```

```
 int u = Q.front (); Q.pop();
```

```
 for (int v : G[u]) {
```

```
 --pred[v];
```

```

```

```
 if (pred[v] == 0) Q.push(v);
```

```
 }
```

```
 return lvl;
```

```
}
```

Cost i justificació:

**Problema 3****(2.25 pts.)**

El president d'una companyia petrolera ens demana ajuda per a decidir en quines ciutats col·locar les gasolineres d'un país molt llunyà. Coneixem el cost d'instal·lar una gasolinera a cada ciutat i també disposem de la llista de totes les carreteres del país, identificades per parells  $\{c_1, c_2\}$ , on  $c_1$  i  $c_2$  són les dues ciutats que uneix aquesta carretera. El nostre objectiu és decidir a quines ciutats cal posar una gasolinera de manera que el cost total d'instal·lació sigui mínim i que, per a tota carretera  $\{i, j\}$  hi hagi una gasolinera a  $i$ , a  $j$  o en ambdues ciutats.

- a) (2 pts) Completeu els buits en el programa següent perquè resolgui el problema anterior. Les ciutats s'identifiquen amb nombres consecutius 0, 1, 2, etc.

```
struct GasStations {
 int n;
 vector<int> cost;
 vector<vector<int>> roads;
 vector<bool> best_solution;
 int best_cost ;
```

```

void rec(int i, vector<bool>& partial_sol, int partial_cost) {
 if (i == n) {
 best_cost = partial_cost ;
 best_solution = partial_sol ;
 }
 else {
 if (< best_cost) {
 partial_sol [i] = ;
 rec(,partial_sol,);
 }

 bool needed = false;
 for (auto& c : roads[i])
 if (and) needed = true;

 if (not needed) {
 partial_sol [i] = ;
 rec(,partial_sol,);
 } } // tanquem rec

GasStations (const vector<int>& c, vector<vector<int>>& r) {
 n = c.size (); best_cost = INT_MAX;
 this->cost = c; this->roads = r;
 vector<bool> partial_sol(n,false);
 rec (0, partial_sol ,0);
} };

int main (){
 int n; // nombre de ciutats
 cin >> n;
 vector<int> c(n); // cost de cada ciutat
 for (int i = 0; i < n; ++i) cin >> c[i];
 vector<vector<int>> r(n); // carreteres (graf com a llista d'adjacència)
 int c1, c2;
 while (cin >> c1 >> c2) { // Carretera entre c1 i c2
 r[c1].push_back(c2);
 r[c2].push_back(c1);
 }
 GasStations gas(c,r);
 cout << "Best cost: " << gas.best_cost << endl;
 cout << "Chosen cities:";
 for (int i = 0; i < n; ++i)
 if (gas.best_solution [i]) cout << " " << i;
 cout << endl;
}

```

- b) (0.25 pts) Si ara ens canvien el problema lleugerament i ens demanen maximitzar el cost total d'instal·lació, què canviariéu del codi anterior? En cas que considereu que heu de modificar massa coses, podeu explicar a alt nivell quin algorisme implementaríeu.

**Problema 4****(3.25 pts.)**

Recordem el problema de **3-SAT**: una *variable booleana* només pot prendre el valor 0 (fals) o el valor 1 (cert). Un *literal* és una variable booleana  $x$  o la seva negació  $\neg x$ . Una *clàusula* és una disjunció de literals. Una 3-CNF és una conjunció de clàusules que contenen exactament 3 literals. Una assignació  $I$  de valors a les variables booleanes satisfà un literal  $x$  si  $I(x) = 1$ , i satisfà un literal  $\neg x$  si  $I(x) = 0$ . El problema de **3-SAT** consisteix a, donada una 3-CNF  $F$ , determinar si existeix una assignació que satisfà **com a mínim** un literal de cada clàusula. Fins aquí, res de nou.

Introduïm ara el problema de **ONE-IN-THREE-SAT** que consisteix a, donada una 3-CNF  $F$ , determinar si existeix una assignació que satisfà **exactament** un literal de cada clàusula de  $F$ .

- (a) (0.75 pts.) Considereu les 3 clàusules:

$$\begin{array}{l} x_1 \vee \neg x_2 \vee x_3 \\ x_1 \vee x_4 \vee x_5 \\ \neg x_2 \vee x_4 \vee x_5 \end{array}$$

Escriviu una assignació que satisfaci exactament un literal de cada clàusula.

$$I(x_1) = \boxed{\phantom{0}}, I(x_2) = \boxed{\phantom{0}}, I(x_3) = \boxed{\phantom{0}}, I(x_4) = \boxed{\phantom{0}}, I(x_5) = \boxed{\phantom{0}}.$$

Existeix alguna assignació  $I$  que satisfaci exactament un literal de cada clàusula i tal que  $I(x_1) = 1$ ? Justifiqueu la vostra resposta.

Escriviu un conjunt **petit** de clàusules de 3 literals que sigui una entrada positiva de **3-SAT** però no ho sigui de **ONE-IN-THREE-SAT**. Acceptarem conjunts de com a molt 4 clàusules, tot i que amb 3 ja n'hauríeu de fer prou. Justifiqueu la vostra resposta.

- (b) (1.5 pts.) Donada una clàusula  $C$  de 3 literals  $C = l_1 \vee l_2 \vee l_3$ , definim el conjunt de clàusules  $R(C) = \{C_1, C_2, C_3\}$ , on

$$\begin{aligned} C_1 &= \neg l_1 \vee a \vee b \\ C_2 &= l_2 \vee b \vee c \\ C_3 &= \neg l_3 \vee c \vee d \end{aligned}$$

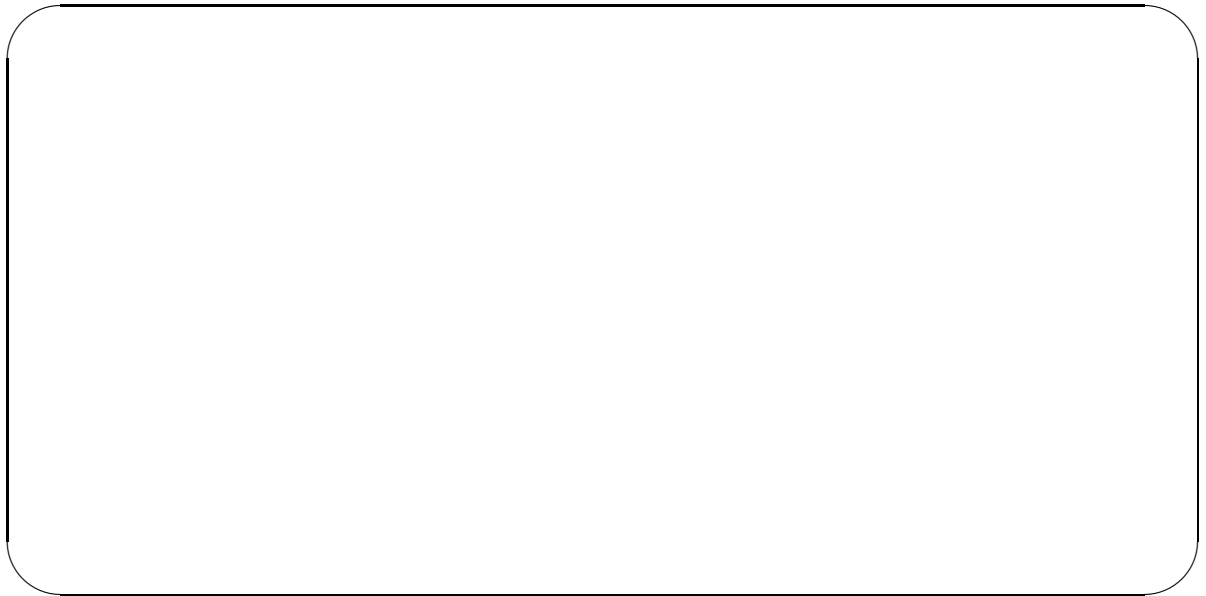
i  $a, b, c, d$  són variables booleanes noves. *Nota:* si  $x$  és una variable i el literal  $l$  és  $\neg x$ , entendrem que  $\neg l$  es correspon a  $x$ .

Demostreu que si  $I$  és una assignació que satisfà exactament un literal de cada clàusula de  $R(C)$ , aleshores  $I$  satisfà almenys un literal de  $C$ .

Demostreu que donada una assignació  $I$  que satisfà almenys un literal de  $C$ , es pot construir una assignació  $I'$  que satisfà exactament un literal de cada clàusula de  $R(C)$  i que coincideix amb  $I$  sobre  $l_1, l_2$  i  $l_3$ .

(c) (0.5 pts.) Demostreu que **ONE-IN-THREE-SAT** és NP-difícil.

(d) (0.5 pts.) Demostreu que **ONE-IN-THREE-SAT** és NP-complet.





**Examen Final EDA Duració: 2h 45min****09/06/2020**

Cada alumne té assignada, de manera aleatòria, l'opció A o B de cada problema.

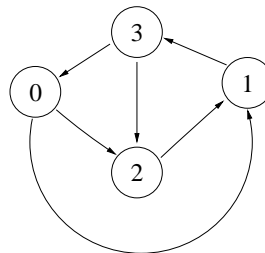
**Problema 1****(2 pts.)**

Test [https://jutge.org/problems/X53101\\_ca](https://jutge.org/problems/X53101_ca) a omplir a través de Jutge.org. Teniu 30 minuts per entregar les respostes.

**Problema 2A****(2 pts.)**

Donat un graf dirigit  $G$ , volem determinar si  $G$  és Hamiltonià: és a dir, si existeix algun cicle que passi exactament una vegada per cada vèrtex. A continuació podeu veure un exemple d'arxiu d'entrada i el graf que representa. L'entrada comença amb una línia de la forma  $n \ m$ , on  $n$  és el nombre de vèrtexs (que es representaran amb nombres entre 0 i  $n - 1$ ) i  $m$  és el nombre d'arcs. A continuació venen  $m$  línies de la forma  $u \ v$  corresponents a arcs  $u \rightarrow v$ .

```
4 6
0 2
0 1
2 1
3 2
1 3
3 0
```



En aquest cas tenim un graf Hamiltonià, ja que  $2 \rightarrow 1 \rightarrow 3 \rightarrow 0 \rightarrow 2$  és un cicle com el que busquem. Ompliu amb una instrucció o expressió els buits del codi que teniu a continuació per tal de determinar si un graf dirigit és Hamiltonià.

```
class HamiltonianGraph {
 vector<vector<int>>> G;
 int n;
 vector<int> s; // solucio parcial en construccio
 vector<bool> used;
 bool found; // si hem trobat un cicle Hamiltonia
 vector<int> sol; // si found, sol guarda el cicle Hamiltonia

 void recursive (){
 int u = s.back ();
 if (s.size () == n) {
 for (int v : G[u]) {
 if () {
 found = true;
 sol = s;
 } } }
 else {
 for (int v : G[u]) {
 if () {
 s.push_back(v);
 used[v] = ;
 recursive ();
 ;
 }
 }
 }
 }
 }
}
```

```

 _____ ;
 if (found) return; } } } }

```

**public:**

```

HamiltonianGraph (vector<vector<int>>> &G) {
 this->G = G;
 n = G.size ();
 used = vector<bool>(n,false);
 s = vector<int>(1,0);
 used[0] = true;
 found = false;
 recursive ();
}

```

```

bool has_solution () {return found;}
vector<int> solution() {return sol;}
};

```

```

int main() {
 int n, m;
 cin >> n >> m;
 vector<vector<int>>> G(n);
 for (int i = 0; i < m; ++i){
 int u, v;
 cin >> u >> v;
 G[u].push_back(v);
 }
 HamiltonianGraph ham(G);
 cout << ham.has_solution () << endl;
}

```

### Problema 2B

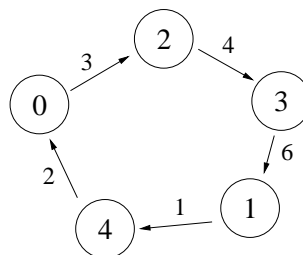
(2 pts.)

Volem resoldre el conegut problema del viatjant de comerç: donat un conjunt de ciutats i les distàncies entre elles cal trobar, d'entre tots els cicles que passin per totes les ciutats exactament una vegada, un que tingui distància total mínima. A continuació podeu veure un exemple d'arxiu d'entrada i un cicle de distància mínima. L'entrada comença amb un enter  $n$ , que indica el nombre de ciutats i a continuació trobem una matriu simètrica de naturals  $n \times n$  amb totes les distàncies.

```

5
0 7 3 9 2
7 0 6 6 1
3 6 0 4 2
9 6 4 0 3
2 1 2 3 0

```



En aquest cas el cicle de distància mínima té distància 16. Ompliu amb una instrucció o expressió els buits del codi que teniu a continuació per tal de trobar un cicle de distància mínima.

```

const int infinite = numeric_limits<int>::max();
class TSP {
 int n;
 vector<vector<int>>> M;
 vector<int> s; // solucio parcial en construccio
 vector<bool> used;
 vector<int> best_sol; // millor cicle trobat fins ara
 int best_cost ; // cost del millor cicle trobat fins ara

```

```

 void recursive (int c) {
 int u = s.back ();
 if (s.size () == n) {
 c += M[u][0];
 if (c < best_cost) {
 best_cost = c;
 best_sol = s;
 }
 }
 else {
 for (int v = 0; v < n; ++v)
 if () {
 if (c + M[u][v] < best_cost) {
 s.push_back(v);
 used[v] = ;
 recursive ();
 ;
 ; } } } }

```

**public:**

```

 TSP (vector<vector<int>>> & M) {
 this->M = M;
 n = M.size ();
 s = vector<int>(1,0);
 used = vector<bool>(n,false);
 used[0] = true;
 best_cost = infinite ;
 recursive (0);
 }

```

```

 vector<int> solution () {return best_sol ;}
 int cost () {return best_cost ;}
};

```

```

int main(){
 int n;
 cin >> n;

```

```

vector<vector<int>>> M(n, vector<int>(n));
for (int i = 0; i < n; ++i)
 for (int j = 0; j < n; ++j)
 cin >> M[i][j];
TSP tsp(M);
cout << tsp.cost() << endl;
}

```

**Problema 3A****(3 pts.)**

Una empresa siderúrgica ens demana ajuda per tal de decidir en quin ordre s'han de produir les peces que tenen encarregades. Afortunadament el problema és fàcil de definir, ja que l'única restricció és que existeixen parelles de peces  $(p_1, p_2)$  tals que  $p_1$  s'ha de produir abans de  $p_2$ . Un arxiu d'entrada té el format següent:

```

5 4
0 1 1 3 2 3 3 4

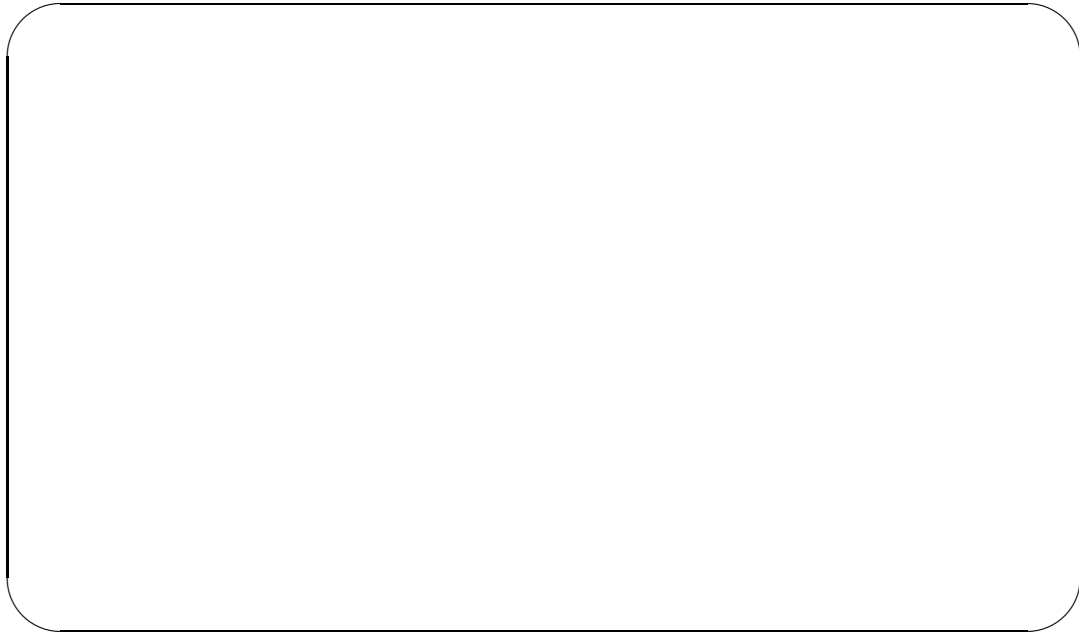
```

La primera línia indica que tenim  $n$  peces (nombres entre 0 i  $n - 1$ ) i  $m$  parelles de peces amb restricció entre sí, en aquest cas 5 peces i 4 parelles. A continuació trobem  $m$  parelles  $p_1 p_2$  que indiquen que la peça  $p_1$  s'ha de produir abans que la peça  $p_2$ . Una possible solució a aquest problema concret és l'ordre: (0,2,1,3,4).

Sabent que tots els arxius d'entrada tindran almenys un ordre possible, considereu la solució al problema donada a l'arxiu `ex-packages.cc`. Veureu que l'arxiu, entre altres coses, conté una implementació d'una cua de prioritats amb algunes operacions addicionals. Quan analitzeu costos, ignoreu les instruccions `assert` i el seu cost.

- (a) (1 pt.) Analitzeu el cost asimptòtic en temps en el cas pitjor de la funció `write_packages` en funció d' $n$  i  $m$ .

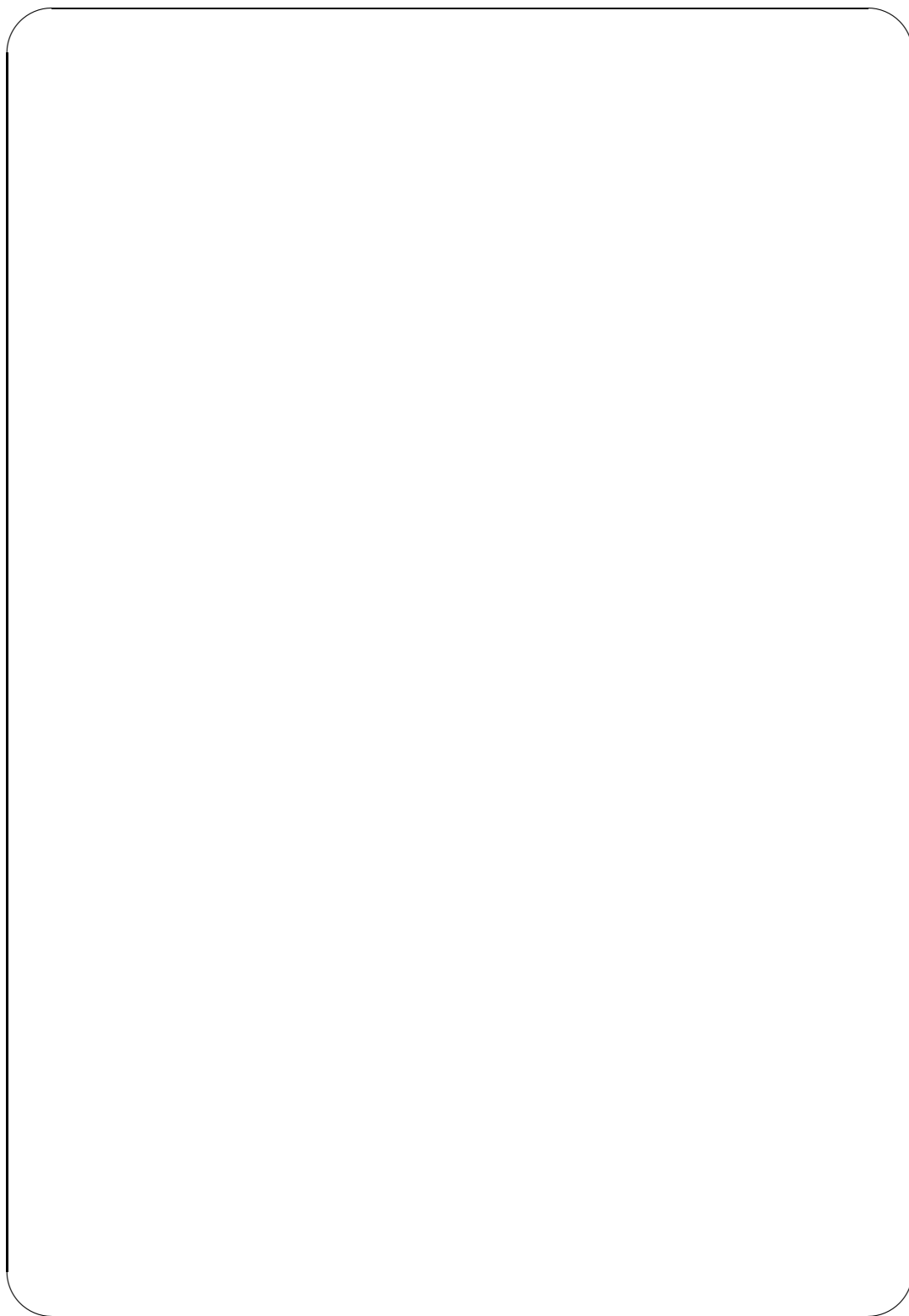
- (b) (1 pt.) Expliqueu en paraules com implementaríeu un programa que millori el temps asimptòtic en el cas pitjor de la solució donada al fitxer `ex-packages.cc`. Justifiqueu el cost de la vostra proposta.



- (c) (1 pt.) És fàcil veure que el problema que estem considerant molt sovint admet múltiples solucions. Per exemple,  $(0, 1, 2, 3, 4)$  també és un ordre correcte per a la instància anterior. De totes les solucions, ens demanen la que sigui lexicogràficament més petita.

Recordem que una solució  $p = (p_1, p_2, \dots, p_n)$  és lexicogràficament més petita que una altra solució  $q = (q_1, q_2, \dots, q_n)$  si existeix un índex  $k$  amb  $1 \leq k \leq n$  tal que per a tota  $j$  tal que  $1 \leq j < k$  es compleix que  $p_j = q_j$  i  $p_k < q_k$ . És a dir, si, vistes com a paraules,  $p$  apareixeria abans que  $q$  al diccionari.

Modifiqueu el codi donat de manera que la solució escrita sigui la lexicogràficament més petita. Indiqueu només les parts modificades, que haurien de ser molt poques.



Arxiu ex-packages.cc:

```

#include<iostream>
#include<vector>
#include<assert.h>

using namespace std;

template <typename Elem, typename Key>
class PriorityQueue {
private:
 vector<pair<Elem,Key>> v; // Table for the heap (position 0 is not used)
public:
 // Constructor. Creates an empty priority queue.
 PriorityQueue () {
 v.push_back({Elem(),Key()});
 }

 // Inserts a new element.
 // PRE: x.first does not belong to the priority queue
 void insert (const pair<Elem,Key> & x) {
 assert (not contains (x.first));
 v.push_back(x);
 shift_up (size ());
 }

 // Removes and returns the minimum element.
 pair<Elem,Key> remove_min () {
 if (empty()) throw "Priority queue is empty";
 pair<Elem,Key> x = v[1];
 v[1] = v.back ();
 v.pop_back ();
 shift_down (1);
 return x;
 }

 // Returns the minimum element
 pair<Elem,Key> minimum () {
 if (empty()) throw "Priority queue is empty";
 return v[1];
 }

 // Returns the size of the priority queue.
 int size () {
 return v.size () - 1;
 }

 // Indicates if the priority queue is empty.
 bool empty () {
 return size () == 0;
 }
};

```

```

}

// Pre: x is an element in the priority queue
// the key of x is \neq newKey
// Post: the key of x is updated to newKey
void decrease_key (const Elem& x, const Key& newKey) {
 assert (contains(x));
 int idx = find(x);
 v[idx].second = newKey;
 shift_up (idx);
}

```

```

// Returns whether x belongs to the priority queue
bool contains(const Elem& x) {
 int idx = find(x);
 return idx \neq -1;
}

```

```

// PRE: x belongs to the priority queue
// Returns the key of the element x
Key key(const Elem&x) {
 assert (contains(x));
 return v[find(x)].second;
}

```

**private:**

```

// Returns the idx in v of the element x
// Returns -1 if not present
int find (const Elem& x) {
 for (uint i = 1; i < v.size (); ++i) if (v[i].first == x) return i;
 return -1;
}

```

```

//Shifts a node up in the tree, as long as needed.
void shift_up (int i) {
 if (i \neq 1 and v[i/2].second > v[i].second) {
 swap(v[i], v[i/2]);
 shift_up (i/2);
 }
}

```

```

//Shifts a node down in the tree, as long as needed.
void shift_down (int i) {
 int n = size ();
 int c = 2*i;
 if (c \leq n) {
 if (c+1 \leq n and v[c+1].second < v[c].second) c++;
 if (v[i].second > v[c].second) {
 swap(v[i], v[c]);

```



```

 shift_down(c);
 } } }

};

void write_packages (const vector<vector<int>>& G) {

 int n = G.size ();

 vector<int> indegree(n,0);
 for (int u = 0; u < n; ++u)
 for (int v:G[u]) ++indegree[v];

 PriorityQueue<int,int> Q;
 for (int u = 0; u < n; ++u) Q.insert({u,indegree[u]});

 while (not Q.empty()) {
 pair<int,int> p = Q.remove_min();
 assert (p.second == 0);
 cout << p.first << " ";
 for (int v : G[p.first]) {
 int d = Q.key(v);
 Q.decrease_key (v,d-1);
 }
 }
 cout << endl;

}

int main(){
 int n, m;
 cin >> n >> m;
 vector<vector<int>>> G(n);

 for (int i = 0; i < m; ++i) {
 int x, y;
 cin >> x >> y;
 G[x].push_back(y);
 }
 write_packages (G);
}

```

**Problema 3B****(3 pts.)**

De ben segur que esteu familiaritzats amb l'algorisme de Dijkstra. En aquest cas, considerem que estem treballant amb un graf dirigit amb pesos i que volem saber la distància entre dos nodes. L'entrada que ens proporcionen segueix el format següent:

```
3 3
0 2 99 0 1 40 1 2 60
0 2
```

La primera línia indica que tenim  $n$  nodes (nombres entre 0 i  $n - 1$ ) i  $m$  arestes. A continuació trobem  $m$  arestes representades per tripletes  $u\ v\ c$  indicant que hi ha un arc de  $u$  cap a  $v$  amb pes  $c$ . Finalment trobem una parella de vèrtexs  $x$  i  $y$  que ens indica que volem saber la distància de  $x$  a  $y$ .

Aquesta vegada no heu de solucionar aquest problema, sinó que heu d'analitzar i millorar la solució que us donem a l'arxiu `ex-dijkstra.cc`. Quan analitzeu costos, ignoreu les instruccions `assert` i el seu cost.

- (a) (1 pt.) Responen a les preguntes següents tot omplint els blancs. Les respostes només poden dependre de  $n$  i  $m$ , excepte a la primera pregunta, cal donar sempre una justificació.

Quin és el cost de la funció `find`?  $O(\text{ })$

Quantes vegades s'insereix, com a molt, cada vèrtex a la cua de prioritats  $Q$ ?

Quantes vegades es treu, com a molt, cada vèrtex de la cua de prioritats  $Q$ ?

Quantes vegades es crida, com a molt, a la funció `decrease_key`?

Quin és el cost asimptòtic en temps, en el cas pitjor, de la funció `dijkstra`?

- (b) (1.5 pts.) Modifiqueu la solució que us hem donat per tal que la funció *find* tingui, en el cas pitjor, cost asimptòtic  $\Theta(1)$ . Podeu assumir en la resta d'aquest problema que les cerques en un *unordered\_map* sempre tenen cost constant. Pista: podeu modificar altres funcions a més de *find*.

Escriviu només les parts que s'han de modificar.

- (c) (0.5 pts.) Assumint que *find* té temps constant, quin és el cost en temps de la nova solució? Si sabem que  $m = \Theta(n^2)$ , com es compara asimptòticament el cost en temps de la funció *dijkstra* en la nova solució respecte al cost en la solució inicial que us hem donat?

Arxiu ex-dijkstra.cc:

```
#include<iostream>
#include<vector>
#include<unordered_map>
#include<limits>
#include<assert.h>

using namespace std;

const int infinite = numeric_limits<int>::max();

template <typename Elem, typename Key>
class PriorityQueue {
private:
 vector<pair<Elem,Key>> v; // Table for the heap (position 0 is not used)
public:
 // Constructor. Creates an empty priority queue.
 PriorityQueue () {
 v.push_back({Elem(),Key()});
 }

 // Inserts a new element.
 //PRE: x.first does not belong to the priority queue
 void insert (const pair<Elem,Key> & x) {
 assert (not contains (x.first));
 v.push_back(x);
 shift_up (size ());
 }

 // Removes and returns the minimum element.
 pair<Elem,Key> remove_min () {
 if (empty()) throw "Priority queue is empty";
 pair<Elem,Key> x = v[1];
 v[1] = v.back ();
 v.pop_back ();
 shift_down (1);
 return x;
 }

 // Returns the minimum element.
 pair<Elem,Key> minimum () {
 if (empty()) throw "Priority queue is empty";
 return v[1];
 }

 // Returns the size of the priority queue.
 int size () {
 return v.size () - 1;
 }
}
```

```

//Indicates if the priority queue is empty.
bool empty () {
 return size () == 0;
}

// Pre: x is an element in the priority queue
// the key of x is < newKey
// Post: the key of x is updated to newKey
void decrease_key (const Elem& x, const Key& newKey) {
 assert (contains (x));
 int idx = find (x);
 v[idx].second = newKey;
 shift_up (idx);
}

// Returns whether x belongs to the priority queue
bool contains(const Elem& x) {
 int idx = find (x);
 return idx ≠ -1;
}

// PRE: x belongs to the priority queue
// Returns the key of the element x
Key key(const Elem&x){
 assert (contains (x));
 return v[find (x)].second;
}

private:

// Returns the idx in v of the element x
// Returns -1 if not present
int find (const Elem& x) {
 for (uint i = 1; i < v.size (); ++i) if (v[i].first == x) return i;
 return -1;
}

//Shifts a node up in the tree, as long as needed.
void shift_up (int i) {
 if (i ≠ 1 and v[i/2].second > v[i].second) {
 swap(v[i], v[i/2]);
 shift_up (i/2);
 }
}

//Shifts a node down in the tree, as long as needed.
void shift_down (int i) {
 int n = size ();
 int c = 2*i;

```

```

 if (c ≤ n) {
 if (c+1 ≤ n and v[c+1].second < v[c].second) c++;
 if (v[i].second > v[c].second) {
 swap(v[i], v[c]);
 shift_down(c);
 } } }

};

int dijkstra (const vector<vector<pair<int,int>>>& G, int x, int y) {
 int n = G.size ();
 vector<bool> removed(n,false);
 PriorityQueue<int,int> Q;
 Q.insert ({x,0});

 while (not Q.empty()) {
 pair<int,int> m = Q.remove_min();
 int u = m.first ;
 int d_u = m.second;
 assert (not removed[u]);
 removed[u] = true;
 if (u == y) return d_u;
 for (const pair<int,int>& p : G[u]) {
 int v = p.first ;
 int c_v = p.second;
 if (not removed[v]){
 if (not Q.contains(v)) Q.insert ({v, d_u + c_v });
 else if (Q.key(v) > d_u + c_v) Q.decrease_key (v, d_u + c_v);
 }
 }
 }
 return -1;
}

int main(){
 int n, m;
 int c = 0;
 while (cin >> n >> m) {
 ++c;
 vector<vector<pair<int,int>>> G(n); //pairs are (vertex,weight)
 for (int i = 0; i < m; ++i) {
 int u, v, c;
 cin >> u >> v >> c;
 G[u].push_back({v,c});
 }
 int x, y;
 cin >> x >> y;

 int d = dijkstra (G,x,y);
 }
}

```

```

 if (d == -1) cout << "no path from " << x << " to " << y << endl;
 else cout << d << endl;
 }
}

```

**Problema 4A****(3 pts.)**

Considereu els dos problemes decisionals següents:

**POBLES**: donats

- un conjunt de pobles  $P$ ,
- un conjunt de camins entre ells  $C = \{c_1, c_2, \dots, c_m\}$ , on cada camí  $c_i$  és una parella de pobles diferents,
- i un natural  $k$ ,

volem saber si existeix un subconjunt  $S \subseteq P$  de mida  $k$  tal que per a tot  $u, v \in S$  amb  $u \neq v$  es té que  $\{u, v\} \in C$ . És a dir, volem saber si existeix un subconjunt de  $k$  pobles tals que estan tots ells connectats dos a dos per un camí directe.

**ENEMISTATS**: donats

- un conjunt de treballadors  $T$ ,
- un conjunt d'enemistats entre ells  $E = \{e_1, e_2, \dots, e_q\}$  on cada enemistat  $e_i$  és una parella de treballadors diferents,
- i un natural  $r$ ,

volem saber si existeix un subconjunt  $M \subseteq T$  de mida  $r$  tal que per a tot  $u, v \in M$  amb  $u \neq v$  es té que  $\{u, v\} \notin E$ . És a dir, volem saber si existeix un conjunt de  $r$  treballadors tals que cap treballador estigui enemistat amb cap altre.

(a) (0.25 pts.) Considereu l'entrada següent per al problema POBLES:

- $P = \{1, 2, 3, 4, 5, 6\}$
- $C = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}, \{3, 6\}\}$
- $k = 4$

És una entrada positiva? Justifiqueu la vostra resposta.

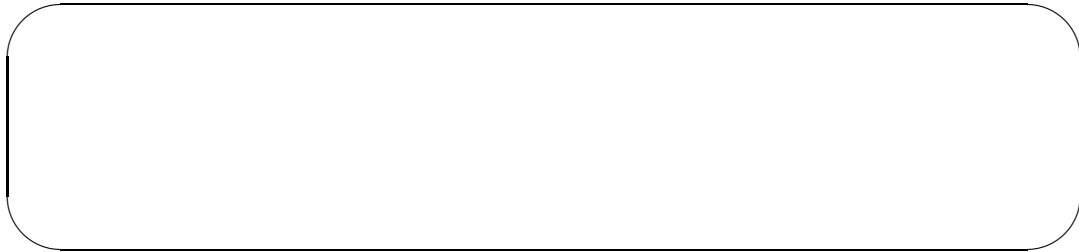
(b) (0.25 pts.) Considereu l'entrada següent per al problema ENEMISTATS:

- $T = \{1, 2, 3, 4, 5\}$
- $E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$




- $r = 3$

És una entrada positiva? Justifiqueu la vostra resposta.



(c) (0.75 pts.) Demostreu que ENEMISTATS  $\in$  NP.



(d) (1.25 pts.) Doneu una reducció polinòmica de POBLES cap a ENEMISTATS i demostreu que és correcta.



- (e) (0.5 pts.) Ens asseguren que POBLES és un problema NP-complet. Podem assegurar que ENEMISTATS és NP-complet? Justifiqueu la resposta.

#### Problema 4B

(3 pts.)

Considereu els dos problemes decisionals següents:

**EQUIP**: donats

- un conjunt de treballadors  $T$ ,
- un conjunt de desitjos de pertinença al mateix equip  $D = \{d_1, d_2, \dots, d_m\}$ , on cada desig  $d_i$  és una parella de treballadors que volen treballar junts,
- i un natural  $k$ ,

volem saber si existeix un subconjunt  $E \subseteq T$  de mida  $k$  tal que per a tot parell  $u, v \in E$  amb  $u \neq v$  es té que  $\{u, v\} \in D$ . És a dir, volem saber si existeix un subconjunt de  $k$  treballadors tals que tots ells han expressat el desig de treballar amb tots els altres membres de l'equip.

**INVESTIGADORS**: donats

- un conjunt d'equips d'investigadors  $I$ ,
- un conjunt de reunions entre parells d'equips  $M = \{m_1, m_2, \dots, m_q\}$ , on cada reunió  $m_i$  és un parell d'equips,
- i un natural  $r$ ,

volem saber si existeix un subconjunt  $S \subseteq I$  de mida  $r$  tal que per a tot parell  $\{u, v\} \in M$ , es compleix que o bé  $u \in S$ , o bé  $v \in S$  o bé tant  $u$  com  $v$  pertanyen a  $S$ . És a dir, volem saber si podem escollir  $r$  equips de manera que en tota reunió hi hagi present almenys un dels equips escollits.

- (a) (0.25 pts.) Considereu l'entrada següent per al problema EQUIP:

- $T = \{1, 2, 3, 4, 5, 6\}$
- $D = \{\{1, 3\}, \{1, 4\}, \{1, 5\}, \{1, 6\}, \{2, 3\}, \{2, 5\}, \{4, 5\}, \{4, 6\}, \{5, 6\}\}$ ,
- $k = 4$

És una entrada positiva? Justifiqueu la vostra resposta.

(b) (0.25 pts.) Considereu l'entrada següent per al problema INVESTIGADORS

- $I = \{1, 2, 3, 4, 5\}$
- $M = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}\},$
- $r = 2$

És una entrada positiva? Justifiqueu la vostra resposta.

(c) (0.75 pts.) Demostreu que INVESTIGADORS  $\in$  NP.

(d) (1.25 pts.) Doneu una reducció polinòmica d'EQUIP cap a INVESTIGADORS i demostreu que és correcta.

- (e) (0.5 pts.) Ens asseguruen que EQUIP és un problema NP-complet. Podem assegurar que INVESTIGADORS és NP-complet? Justifiqueu la resposta.

**Examen Final EDA****Duració: 3h****08/01/2021****Problema 1****(2.5 pts.)**

Donat un vector  $v$  d' $n$  naturals volem determinar si existeix un element *dominant*, és a dir, si existeix un element que apareix més de  $n/2$  vegades. Per exemple:

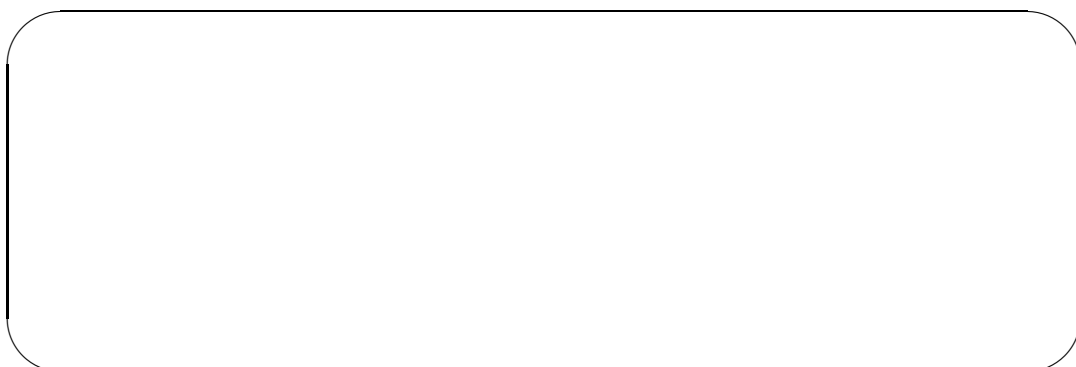
- Si  $v = \{5, 2, 5, 2, 8, 2, 2\}$ , aleshores 2 és l'element dominant perquè apareix  $4 > 7/2$  vegades.
- Si  $v = \{3, 2, 3, 3, 2, 3\}$ , aleshores 3 és l'element dominant perquè apareix  $4 > 6/2$  vegades.
- Si  $v = \{6, 1, 6, 1, 6, 9\}$ , no hi ha cap element dominant perquè cap d'ells apareix més de  $6/2$  vegades.

Volem obtenir una funció en C++ que rebi el vector  $v$  i retorni l'element dominant de  $v$ , o el nombre  $-1$  en cas que no existeixi cap element dominant.

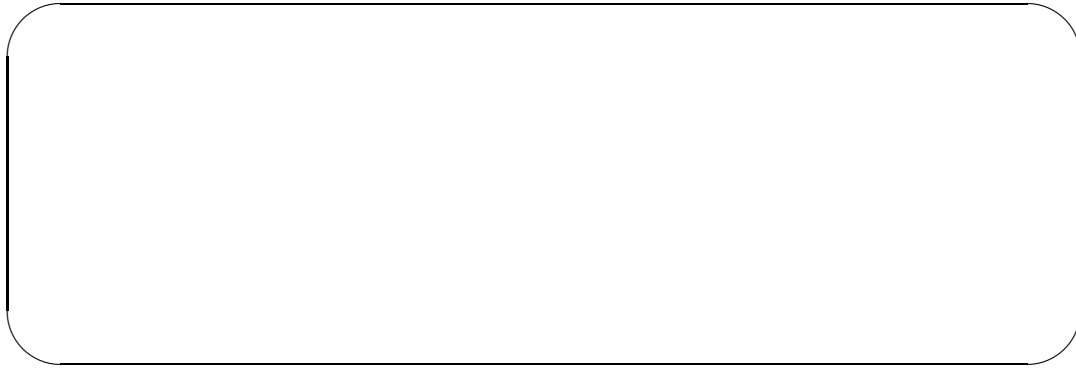
- (a) (1.25 pts.) Un antic estudiant d'EDA llegeix el problema i se n'adona que, si s'utilitzen diccionaris, es pot aconseguir una solució molt neta i prou eficient:

```
int majority_map (const vector<int>& v) {
 map<int,int> M;
 int n = v.size ();
 for (auto& x : v) {
 ++M[x];
 if (M[x] > n/2) return x;
 }
 return -1;
}
```

Si assumim que el *map* s'implementa com un AVL, analitzeu el seu cost en cas pitjor en funció de  $n$ .



Si ara assumim que no hi ha cap element dominant, que enlloc d'un *map* utilitzem un *unordered\_map*, i que aquest s'implementa com una taula de dispersió, quin és el cost del codi en cas mitjà en funció de  $n$ ?



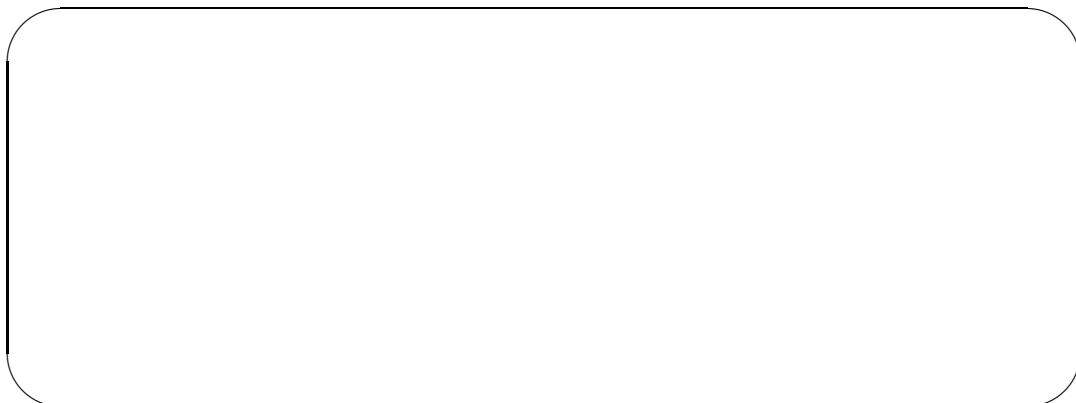
(b) (1.25 pts.) Un estudiant brillant ens proporciona una solució basada en dividir i vèncer.

```
int times (const vector<int>& v, int l, int r, int x) {
 if (l > r) return 0;
 return (v[l] == x) + times(v, l+1, r, x); }

int majority_pairs (const vector<int>& v, int tie_breaker) {
 if (v.size () == 0) return tie_breaker ;
 else {
 int n = v.size ();
 if (n % 2 == 1) tie_breaker = v.back ();
 vector<int> aux;
 for (int i = 0; i < n - 1; i+=2)
 if (v[i] == v[i+1]) aux.push_back(v[i]);
 int cand = majority_pairs (aux, tie_breaker);
 if (cand == -1) return -1;
 int n_times = times(v, 0, n-1, cand);
 if (n_times > n/2 or (2*n_times == n and cand == tie_breaker)) return cand;
 else return -1;
 }
}

int majority_pairs (const vector<int>& v) {
 return majority_pairs (v, -1);
}
```

Analitzeu el cost en cas pitjor, en funció de  $n$ , d'una crida *majority\_pairs*( $v$ ).



**Problema 2****(2 pts.)**

Tal i com passa a les pitjors pel·lícules de sobretaula, la mort d'una tieta llunyana ens deixa una enorme herència d' $M$  milions d'euros. Com a bons nous rics, ens disposem a gastar tot el dineral que ens ha tocat el més ràpid possible. Per a fer-ho obrim el web d'un portal immobiliari i fem una llista de tots els habitatges que ens interessin, amb el seu corresponent preu en milions d'euros.

Finalment, abans d'abandonar per sempre el món de la informàtica decidim escriure un programa que ens indiqui totes les maneres de comprar un subconjunt dels habitatges que ens interessin per **exactament**  $M$  milions d'euros.

Per exemple, si  $M = 10$  i guardem tots els preus dels habitatges en un vector  $p = [4, 2, 6, 2, 3]$ , aleshores el programa ha d'escriure per pantalla  $\{0, 2\}$  i  $\{1, 2, 3\}$ . És a dir, les solucions contenen els índexos en el vector  $p$  dels habitatges que hem de comprar.

(a) (1 pt.) Completa el següent codi perquè resolgui aquest problema:

```
vector<int> p; // prices of the properties
int money; // total money we have to spend

void write_choices (vector<int>& partial_sol, int partial_sum, int idx) {
 if ([] > []) return;
 if ([]) {
 if (partial_sum == money) {
 cout << "{";
 for (int i = 0; i < partial_sol.size(); ++i)
 cout << (i == 0 ? "" : ",") << partial_sol[i];
 cout << "}" << endl;
 }
 }
 else {
 []
 write_choices (partial_sol , [] , []);
 []
 write_choices (partial_sol , [] , []);
 }
}

int main() {
 int n;
 cin >> money >> n;
 p = vector<int>(n);
 for (auto & x : p) cin >> x;
 vector<int> partial_sol;
 write_choices (partial_sol , 0, 0); }
```

(b) (1 pt.) Després d'executar-lo sobre llistes d'habitatges de mida mitjana, ens n'adonem que el programa és massa lent. Expliqueu com implementaríeu algun mecanisme de poda addicional per millorar el comportament del programa. No cal que escriviu codi, però sí que heu de donar prou detalls perquè a partir de la vostra descripció es pugui implementar la poda fàcilment.

**Problema 3****(2.5 pts.)**

Considereu els dos problemes decisionals següents:

**COLORABILITAT:** donats

- un graf  $G$  amb vèrtexs  $V$  i arestes  $E$ ,
- i un natural  $k$ ,

volem saber si existeix una funció  $c : V \rightarrow \{1, 2, \dots, k\}$  de manera que per a tota aresta  $\{u, v\} \in E$  tenim que  $c(u) \neq c(v)$ .

**DISTINCT-ONES:** donats

- un conjunt  $N$  de naturals (potser amb elements repetits),
- i un natural  $p$ ,

volem saber si podem distribuir els naturals de  $N$  en  $p$  conjunts (és a dir, cada nombre ha d'anar a parar a exactament un conjunt), de manera que si dos nombres van a parar al mateix conjunt, no poden tenir cap 1 en la mateixa posició de la seva representació en binari. És a dir,  $8 = 1000_2$  i  $5 = 0101_2$  poden anar al mateix conjunt, però  $3 = 011_2$  i  $6 = 110_2$  no poden anar junts perquè el segon bit dels dos és 1.

(a) (0.5 pts.) Considereu la instància del problema DISTINCT-ONES:

- $N = \{3, 6, 8, 20, 22\}$
- $p = 3$

És una instància positiva? Justifiqueu la vostra resposta.



(b) (1 pt.) Demostreu que DISTINCT-ONES  $\in$  NP.

(c) (1 pt.) Demostreu que la següent reducció de COLORABILITAT a DISTINCT-ONES és una reducció polinòmica correcta:

Donada una instància  $(G, k)$  de COLORABILITAT, on  $G$  té vèrtexs  $V = \{v_1, v_2, \dots, v_n\}$  i arestes  $E = \{e_0, e_1, \dots, e_{m-1}\}$ , construïm la següent instància  $(N, p)$  de DISTINCT-ONES:

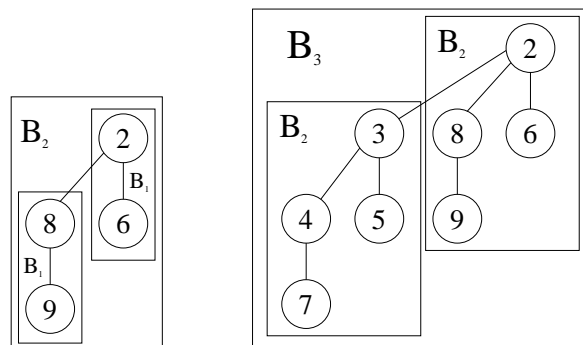
- $N = \{x_1, x_2, \dots, x_n\}$ , on tots els  $x_i$  tenen  $m$  bits i el  $j$ -èssim bit de  $x_i$  és 1 si i només si  $v_i \in e_j$  per a tot  $0 \leq j < m$ . Cada  $x_i$ , doncs, està associat a un vèrtex  $v_i$  de  $G$ .
- $p = k$ .

**Problema 4****(3 pts.)**

Definim  $B_k$ , un arbre binomial d'ordre  $k$  de la manera següent:

- $B_0$  és un arbre format per un únic node.
- Per a tot  $k > 0$ , l'arbre  $B_k$  resulta de prendre un arbre  $B_{k-1}$  amb arrel  $r$  i afegir-hi, com a fill de més a l'esquerra de  $r$ , un altre arbre  $B_{k-1}$ .

En la següent imatge podeu veure com es formen els arbres  $B_2$  i  $B_3$ , respectivament.

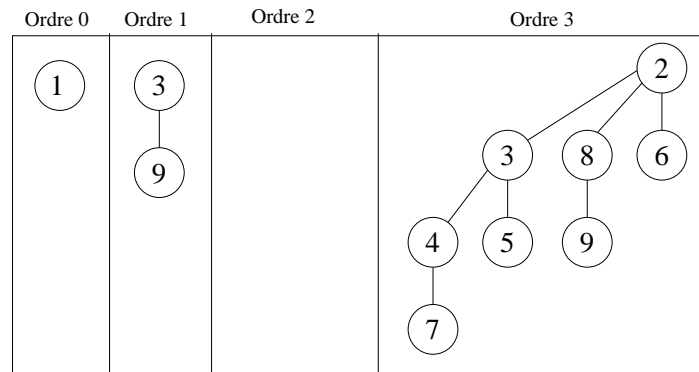


(a) (0.5 pts.) Quants nodes té un arbre  $B_k$ ? Demostrea-ho formalment.

Un *heap binomial* és un conjunt d'arbres binomials que satisfan les següents propietats:

- Cada arbre binomial satisfà la propietat de min-heap: la clau d'un node és major o igual que la clau del seu pare.
- Per a cada  $k \geq 0$ , hi ha com a molt un arbre binomial d'ordre  $k$ .

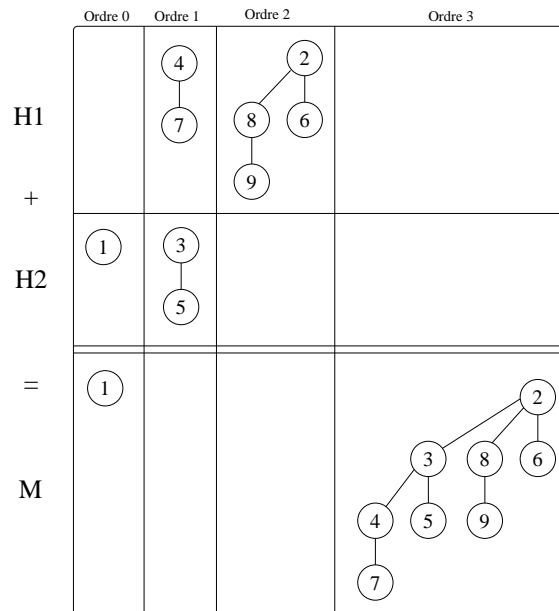
A la següent figura podeu veure un heap binomial amb 11 nodes:



- (b) (0.75 pts.) Volem construir un heap binomial amb  $n$  nodes. Quants arbres binomials de cada ordre tindrà? Exemple: si  $n = 11$  tindrem un arbre binomial d'ordre 0, un d'ordre 1 i un d'ordre 3 (fixeu-vos en la figura superior).

- (c) (0.75 pts.) Donats dos arbres binomials  $A$  i  $B$  d'ordre  $k$  que satisfan la propietat de min-heap, com els podem combinar en temps constant per tal de formar un arbre binomial d'ordre  $k + 1$  que satisfaci la propietat de min-heap i que contingui tots els nodes d' $A$  i  $B$ ?

- (d) (1 pt.) Una de les particularitats dels heaps binomials és que podem fusionar dos heaps binomials que tinguin  $\Theta(n)$  nodes en temps  $\Theta(\log n)$ . L'algorisme s'assembla molt al de la suma en binari: processarem els arbres de menor a major ordre. Per a cada  $k$ , fusionarem els arbres d'ordre  $k$ , considerant que podem tenir un "carry" d'ordre  $k$  de la suma anterior. Teniu un exemple a la següent figura:



Ompliu el següent codi per a fusionar dos heaps binomials que tenen claus enteres:

```

class BinomialHeap {
 class Node {
 public:
 int key;
 vector<Node*> children;
 Node(int k, vector<Node*> c) : key(k), children(c){}
 };
 typedef Node* Tree;
 vector<Tree> roots; // roots[k] is the binomial tree of order k, NULL if none
 BinomialHeap(vector<Tree>& r) : roots(r) {}

 // Given t1, t2 binomial trees of order k that satisfy the min-heap property,
 // returns a binomial tree of order k+1 satisfying the min-heap property that
 // contains all elements of t1 and t2. You can use this function in your code.
 Tree mergeTreesEqualOrder(Tree t1, Tree t2);
 void merge(BinomialHeap& h);
public:
 BinomialHeap(){}
 void push(int k);
 void pop();
 int top(); };

void BinomialHeap::merge (BinomialHeap& h){
 // Make sure vector roots has same size in both heaps (makes code simpler)
 while (h.roots.size() < roots.size()) h.roots.push_back(NULL);

```

```

while (h.roots.size() > roots.size()) roots.push_back(NULL);
vector<Tree> newRoots(roots.size());
Tree carry = NULL;
for (int k = 0; k < roots.size(); ++k) {
 if (roots[k] == NULL and h.roots[k] == NULL) {
 newRoots[k] = ;
 carry = ; }
 else if (roots[k] == NULL) {
 if (carry == NULL) newRoots[k] = ;
 else {
 newRoots[k] = ;
 carry = mergeTreesEqualOrder(,); }
 }
 else if (h.roots[k] == NULL) {
 if (carry == NULL) newRoots[k] = ;
 else {
 newRoots[k] = ;
 carry = mergeTreesEqualOrder(,); }
 }
 else {
 newRoots[k] = ;
 carry = mergeTreesEqualOrder(,); }
 }

 if (carry != NULL) newRoots.push_back();
 roots = newRoots;
}

```

Raoneu per què, si els dos heaps tenen  $\Theta(n)$  elements, aquesta funció triga temps  $\Theta(\log n)$ .

## Examen Final EDA

Duració: 3h

07/06/2021

## Problema 1

(3 pts.)

Responen a les següents preguntes:

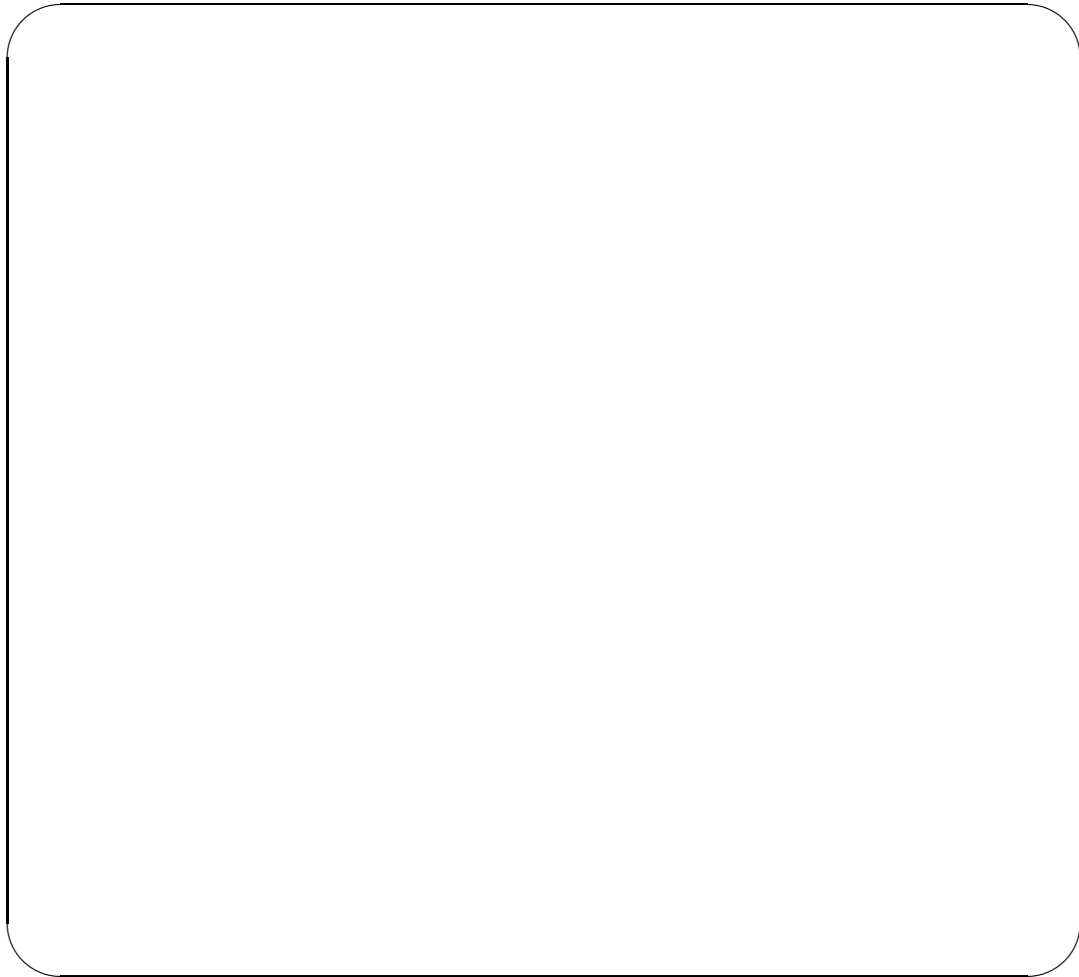
- (a) (1 pt.) Considereu les funcions següents:  $n \log(n^2)$ ,  $n^2$ ,  $n \log n$ ,  $n(\log n)^2$ . Ordeneu-les, de més petita a més gran, segons el seu creixement asimptòtic. Si dues o més funcions tenen el mateix grau de creixement, indiqueu-ho. No cal que raoneu la vostra resposta.

- (b) (1 pt.) Raoneu si l'afirmació següent és certa, falsa o no se sap: "Hi ha més problemes NP-complets que pertanyen a  $P$  que problemes NP-complets que no hi pertanyen".

- (c) (1 pt.) Considereu el codi següent (que no calcula res en particular):

```
int a;
int f (const vector<int>& v, int l, int r) {
 if (l ≥ r) return 0;
 int tot = 0;
 int m = (l+r)/2;
 for (int i = 0; i < a; ++i) {
 if (i%2 == 0) tot += f(v, l, m);
 else tot += f(v, m+1, r);
 }
 for (int i = l; i ≤ r; ++i)
 for (int j = i+1; j ≤ r; ++j)
 tot += v[i]*v[j];
 return tot;
}
```

Si  $v$  és un vector de mida  $n$ , determineu de forma raonada el cost d'una crida  $f(v, 0, n - 1)$  en funció d' $n$  segons el valor de la constant  $a \geq 0$ , que és un enter que no canvia durant l'execució del programa.

**Problema 2****(2.5 pts.)**

Un *graf signat* és un graf **no dirigit** i **connex** on cada aresta té una etiqueta  $+1$  o  $-1$ . S'utilitzen àmpliament en psicologia i ciències socials per a modelitzar les relacions entre persones, grups, corporacions, nacions, etc.

Diem que un graf signat és *estable* si tot cicle del graf té signe positiu. El signe d'un camí o un cicle en el graf és el producte de les etiquetes de les arestes que formen el camí o cicle. Per exemple, un triangle amb tres arestes positives, o amb dues arestes negatives i una positiva és estable, mentre que un triangle amb tres arestes negatives o dues positives i una negativa no ho és.

Un teorema clàssic de la teoria de grafs signats estableix que un graf signat és estable si i només si el conjunt de vèrtexs es pot dividir en dos equips  $A$  i  $B$ , possiblement buits, de manera que tota aresta positiva connecta vèrtexs del mateix equip, i tota aresta negativa connecta vèrtexs d'equips oposats.

- a) (2 pts.) Ompliu les caselles del codi següent per tal de determinar si un graf signat  $G$  és o no estable.

```

const int A = 0; const int B = 1; const int no_team = -1;

struct edge {
 int target; // id of other vertex
 int sign; // -1 or +1
};

typedef vector<vector<edge>> signed_graph;

int opposed_team(int b) {return (b == A ? B : A);}

int expected_team(int t, int sign) {
 if (sign == 1) return t;
 else return opposed_team(t);
}

bool is_stable (const signed_graph& G, int u, vector<int>& team) {
 for (auto& e : G[u]) {
 int e_t = expected_team(team[u], e.sign);
 if () return false;
 if (team[e.target] == no_team) {
 ;
 if () return false;
 }
 }
 return ;
}

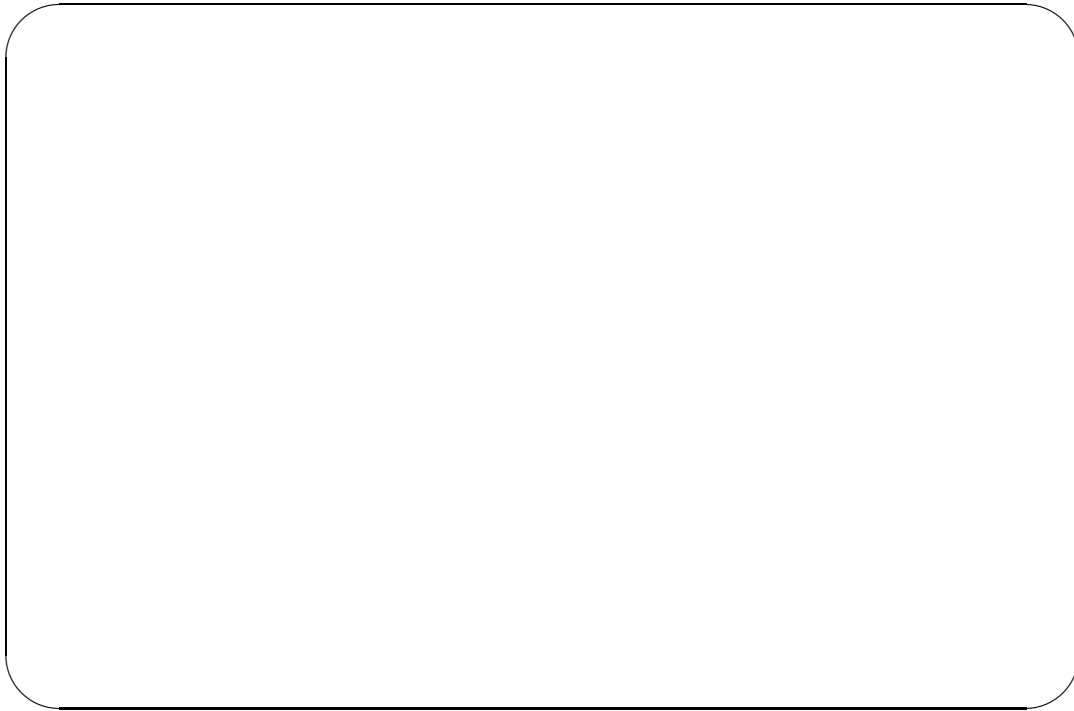
bool is_stable (const signed_graph& G, vector<int>& team) {
 team[0] = A;
 return is_stable (G, 0, team);
}

int main() {
 int n, m;
 cin >> n >> m;
 signed_graph G(n);
 for (int i = 0; i < m; ++i) {
 int v1, v2, sign;
 cin >> v1 >> v2 >> sign;
 G[v1].push_back({v2, sign});
 G[v2].push_back({v1, sign});
 }
 vector<int> team(n, no_team);
 cout << is_stable (G, team) << endl;
}

```



- b) (0.5 pts). Per què hem assignat l'equip  $A$  al vèrtex 0 i ni tan sols hem provat l'altra possibilitat?



### Problema 3

(2.25 pts.)

En una empresa extraordinàriament jeràrquica i paranoica, tenim  $n$  treballadors identificats amb els nombres  $\{1, 2, \dots, n\}$ . Aquests nombres també determinen l'ordre jeràrquic dels treballadors. Més concretament, si  $i < j$  aleshores  $i$  és superior jeràrquicament a  $j$ .

Els treballadors només poden transmetre missatges a col·legues que siguin inferiors jeràrquicament. No obstant, s'han d'utilitzar canals de transmissió extraordinàriament segurs i amb un alt cost. L'empresa no vol crear  $\Theta(n^2)$  canals de transmissió, sinó que com a molt en vol  $O(n \log n)$ . Un conjunt de canals de transmissió és *vàlid* si, per a tot parell de treballadors  $(i, j)$  amb  $i < j$  o bé existeix un canal  $i \rightarrow j$  o bé existeix un altre treballador  $k$  (amb  $i < k < j$ ) que pot actuar com a intermediari. És a dir, existeixen canals  $i \rightarrow k$  i  $k \rightarrow j$ . Recalquem que no es permeten dos o més salts d'intermediaris. És a dir,  $i \rightarrow k, k \rightarrow l, l \rightarrow j$  no és suficient perquè  $i$  es pugui comunicar de forma vàlida amb  $j$ .

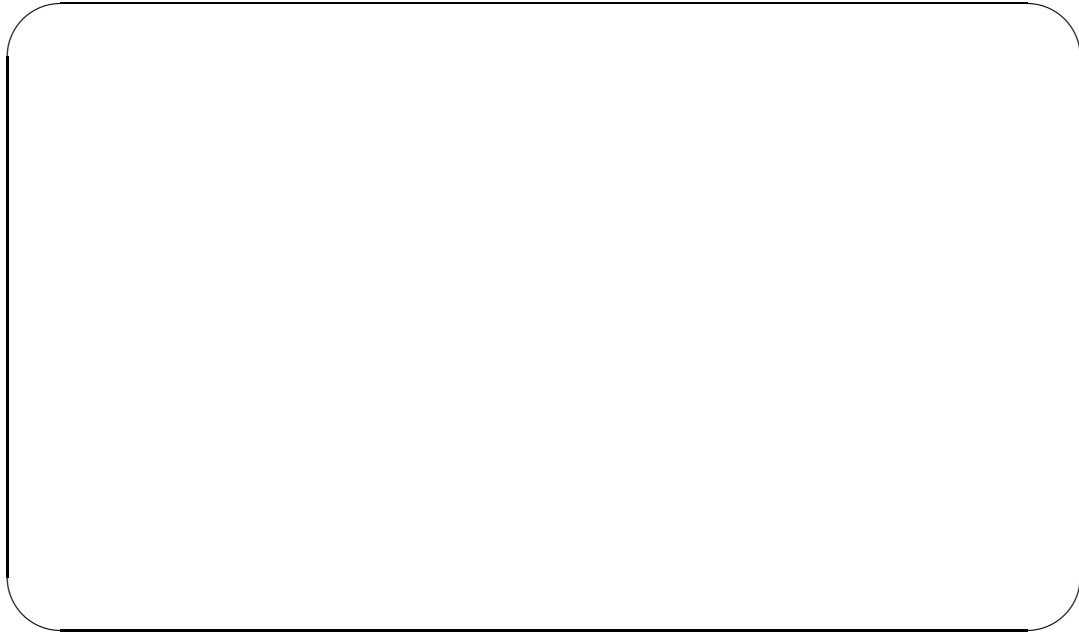
Per exemple, si  $n = 8$ , una possible solució és:

$$\begin{array}{cccccc} 1 \rightarrow 2 & 2 \rightarrow 3 & 7 \rightarrow 8 & 5 \rightarrow 6 & 6 \rightarrow 7 & \\ 6 \rightarrow 8 & 1 \rightarrow 4 & 2 \rightarrow 4 & 3 \rightarrow 4 & 4 \rightarrow 5 & \\ 4 \rightarrow 6 & 4 \rightarrow 7 & 4 \rightarrow 8 & & & \end{array}$$

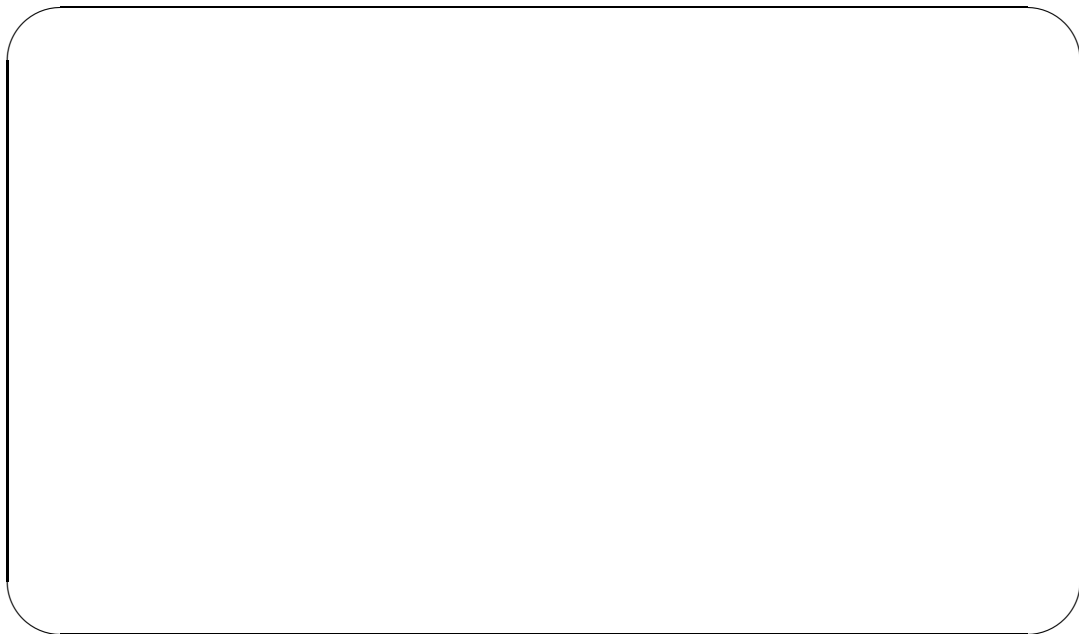
Si 5 vol parlar amb 8 haurà d'utilitzar 6 com a intermediari. En canvi, si 4 vol parlar amb 7 ho podrà fer directament.

- (a) (1.5 pts.) Dissenyeu un algorisme que determini un conjunt vàlid de canals de transmissió de mida com a molt  $O(n \log n)$ . No cal que doneu codi, amb una descripció d'alt nivell serà suficient.

(Pista: partiu els treballadors en dues meitats. Considereu tres tipus de transmissions: entre treballadors de la primera part, entre treballadors de la segona part i entre treballadors de parts diferents.)



- (b) (0.75 pts.) Raoneu per què l'algorisme anterior construeix com a molt  $O(n \log n)$  canals de transmissió.

**Problema 4****(2.25 pts.)**

Volem comprar un cotxe i el fabricant ens ofereix un conjunt d' $n$  equipaments  $E = \{1, 2, \dots, n\}$  que hi podem instal·lar. Hi ha un conjunt d'equipaments  $V \subseteq E$  que volem i un conjunt d'equipaments  $N \subseteq E$  que no volem de cap de les maneres. Evidentment,  $V \cap N = \emptyset$ . Malauradament, el fabricant ens imposa dos tipus de restriccions. D'una banda, hi ha un conjunt  $I \subseteq E \times E$  de parelles d'equipaments que no es poden instal·lar alhora. D'altra banda, hi

ha un conjunt de relacions entre equipaments  $R \subseteq E \times E$ . Una parella  $(i, j) \in R$  significa que si volem instal·lar  $i$  també cal instal·lar  $j$ . Necessitem decidir si és possible satisfer les restriccions imposades pels conjunts  $V, N, I$  i  $R$ . Per exemple, si  $n = 8$  i les restriccions són:

- $V = \{1, 4\}$
- $N = \{2, 3\}$
- $I = \{(1, 3), (2, 3), (1, 5), (5, 8)\}$
- $R = \{(1, 6), (4, 1), (6, 7)\}$

aleshores les restriccions es poden satisfer tot instal·lant, per exemple, els equipaments  $\{1, 4, 6, 7, 8\}$ . Notem que no és el conjunt mínim d'equipaments que satisfà totes les restriccions.

- (a) (1.5 pts.) Construeix una reducció polinòmica del problema que acabem d'introduir cap a  $2SAT$  (determinar si una  $CNF$  en lògica proposicional on cada clàusula té **com a molt** 2 literals és satisfactible). No cal demostrar que la reducció compleix les propietats desitjades.

*Pista:* utilitza variables Booleanes  $p_i$ , amb  $1 \leq i \leq n$ , que signifiquen que l'equipament  $i$  s'instal·larà.

- (b) (0.75 pts.) Podem afirmar que el problema dels equipaments es pot resoldre en temps polinòmic? Només valorarem de forma positiva argumentacions concises i precises.

Examen Final EDA

Duració: 3h

10/01/2022

## Problema 1

(2.5 pts.)

Per simplificar l'anàlisi, en tot aquest problema podeu assumir que una crida al mètode *insert* de la classe *unordered\_set* té sempre cost  $\Theta(1)$ .

- (a) (1 punt) Recordem que la classe *string* en C++ té un mètode *substr(int i, int l)* tal que donat un *string* *s*, la crida *s.substr(i, l)* retorna l'*string* que comença a la posició *i* i té llargada *l*. Assumirem que el cost d'una crida a aquest mètode és  $\Theta(l)$ . Per exemple, si *s* és "paraula", aleshores *s.substr(1,4)* retorna *arau*. Considereu el codi següent:

```
void mystery(const string& s, unordered_set<string>& res){
 res.insert(s);
 if (s.size() > 1) { // call to size has cost Theta(1)
 mystery(s.substr(1, s.size()-1), res);
 mystery(s.substr(0, s.size()-1), res);
 }
}

int main(){
 string s; cin >> s;
 unordered_set<string> res;
 mystery(s, res);
 for (string x : res) cout << x << endl;
}
```

Si l'*string* que es llegeix al *main* és "pep", quants *strings* s'escriuran per la sortida estàndard? Quins són aquests *strings*? No cal raonar la resposta.

Si dins el *main* es llegeix un *string* de mida *n*, quin cost té la crida a *mystery*?

(b) (1 punt) Considerem ara una nova funció *mystery*:

```
void mystery(const string& s, unordered_set <string>& res) {
 for (int i = 0; i < s.size (); ++i)
 for (int j = i; j < s.size (); ++j)
 res . insert (s . substr (i ,j -i+1));
}
```

Si dins el *main* es llegeix un *string* de mida  $n$ , quin cost té ara la crida a *mystery*?

(c) (0.5 pts.) Ompliu el següent codi perquè calculi el mateix que l'apartat anterior:

```
void mystery(const string& s, unordered_set <string>& res){
 for (int k = 1; k ≤ s.size (); ++k)
 for ()
 res . insert (s . substr (i ,k));
}
```

**Problema 2****(2.5 pts.)**

Després de molts anys, el professorat d'EDA decideix renovar el funcionament del Joc. A partir d'ara, les partides seran entre 3 jugadors. El resultat de cada partida és una tripleta  $(j_1, j_2, j_3)$  que indica que el jugador  $j_1$  ha guanyat la partida,  $j_2$  ha estat el segon, i  $j_3$  ha estat el pitjor jugador. Enlloc de les clàssiques rondes, on a cada ronda s'eliminava un jugador, ara es realitzaran un munt de partides entre tripletes de jugadors i en guardarem els resultats. Finalment, per determinar la nota del joc volem establir un rànquing de jugadors, és a dir, una llista ordenada de jugadors on els millors jugadors haurien de sortir a les primeres posicions. Per tal que cap estudiant se senti agreujat, volem garantir que per tot parell de jugadors  $j_1$  i  $j_2$ , si  $j_1$  ha quedat en millor posició que  $j_2$  en alguna partida, aleshores  $j_1$  ha d'aparèixer abans que  $j_2$  al rànquing.

a) (0.75 pts.) Ompliu les caselles del codi següent per trobar un rànquing correcte.

```

struct Match {
 int first ; int second; int third ;
 Match(int f, int s, int t): first (f), second(s), third (t){}
};
int n;
vector<Match> matches;
bool good_ranking (const vector<int>& pos_in_ranking) {

}

bool find_ranking (vector<int>& ranking, vector<int>& pos_in_ranking,
 vector<bool>& used, int idx){
 if (idx == n) return good_ranking(pos_in_ranking);
 else {
 for (int i = 0; i < n; ++i) {
 if (not used[i]) {
 ranking[] = ;
 pos_in_ranking[] = ;
 used[i] = true;
 if (find_ranking (ranking, pos_in_ranking, used, idx+1)) return true;
 used[i] = false;
 } } }
 return false; }

int main () {
 cin >> n; // Students are numbers from 0 to n-1
 int f, s, t; // Read results of matches
 while (cin >> f >> s >> t) matches.push_back(Match(f,s,t));
 vector<int> ranking(n), pos_in_ranking (n);

```

```
vector<bool> used(n,false);
bool b = find_ranking (ranking, pos_in_ranking ,used ,0);
cout << "Ranking found: " << b << endl;
if (b) print_vector (ranking);
}
```

- b) (0.5 pts). Assumim que tenim  $m$  partides i que ens donen una mala implementació de *good\_ranking* que sempre triga temps  $\Theta(m)$ . En funció d' $n$ , quantes vegades es crida a la funció *good\_ranking* en el cas pitjor? A partir d'aquest nombre dóna una fita inferior en funció d' $n$  i  $m$  del cost en cas pitjor del codi anterior. Valorarem la precisió d'aquest fita.

- c) (1.25 pts.) És possible solucionar el problema anterior en temps polinòmic en  $n$  i  $m$  en cas pitjor? Si és possible, explica molt breument com ho faries i justifica el cost. Si no és possible, explica per què.

**Problema 3****(2.5 pts.)**

- (a) (0.75 pts.) Després d'una llarga vida dedicada a obscurs negocis, el cap d'una perillosa organització mafiosa decideix reunir, a mode de comiat, els seus  $n$  col·laboradors per agrair-los la feina feta. No obstant, treballar en assumptes tan delicats ha fet que cadascun d'ells tingui una llista de col·laboradors amb qui no vol coincidir. Sabem, a més, que si  $A$  apareix a la llista de persones que  $B$  vol evitar,  $B$  apareix també a la llista de

persones que  $A$  vol evitar. El cap disposa de 5 dies, i vol citar cada treballador exactament un dia de manera que es respectin els desitjos de no-coincidència. Seríeu capaços de determinar, en temps polinòmic en  $n$ , si es poden organitzar aquestes 5 trobades?

*Nota:* en aquesta pregunta i les següents, **cal** justificar les respostes escrivint reduccions i utilitzant que, per certs problemes que hem vist a classe, es coneixen (o no) algorismes polinòmics que els resolen. No cal demostrar que les reduccions són correctes, però heu de deixar clar des de quin problema a quin altre es fa la reducció.

- (b) (1 pt.) Després de pensar-s'ho una estona, el cap decideix que només vol dedicar 2 dies per reunir a tothom. Seríeu capaços de solucionar aquest problema en temps polinòmic en  $n$ ?

- (c) (0.75 pts.) Finalment, el cap canvia de parer i decideix que les restriccions dels col·laboradors no tenen massa sentit i per tant, les ignorarà. Continua disposant només de 2 dies, i no vol que s'agrupin tots els col·laboradors més importants un dia, i els de menys importància l'altre. Per tal d'aconseguir-ho, sap el patrimoni de tots els seus col·laboradors i vol aconseguir que el patrimoni total dels col·laboradors reunits el primer dia sigui igual al patrimoni dels reunits el segon dia. Seríeu capaços de solucionar aquest problema en temps polinòmic en  $n$ ?



**Problema 4****(2.5 pts.)**

Donat un vector  $v$  d' $n$  enters diferents ordenats de forma creixent i un enter  $x$ , volem determinar si  $x$  apareix a  $v$ . Si és el cas, també volem saber la seva posició. Com bé sabem, aquest problema el podem solucionar en temps  $\Theta(\log n)$  en cas pitjor. No obstant, ens asseguren que de fet  $x$  sempre hi apareix i que gairebé sempre ho fa en les primeres posicions del vector. Amb aquesta informació a les mans, ens interessa trobar un algorisme tal que, si l'aparició d' $x$  dins  $v$  és a la posició  $i$ , aleshores l'algorisme triga temps  $\Theta(\log i)$  en cas pitjor.

- (a) (1 pt.) Completa el següent codi per resoldre, en temps  $\Theta(\log i)$ , el problema que acabem de presentar.

```
// Si x apareix dins v[l...r] retorna i tal que v[i] = x
// Si x no apareix dins v[l...r] retorna -1
// Cost: Theta(log(r-l+1))
int bin_search (int x, const vector<int>& v, int l, int r);

int search (int x, const vector<int>& v) {
 int n = v.size ();
 if (n == 0) return -1;
 int b = 1;
 while ([] and []) b *= 2;
 return bin_search (x, v, b/2, []);
}
```

- (b) (1.5 pts.) Demuestra que, efectivament, si l'aparició d' $x$  dins  $v$  és a la posició  $i$ , aleshores la funció `search` triga temps  $\Theta(\log i)$  en cas pitjor.



---

## Solucions d'Exàmens Parcial

## Solució de l'Examen Parcial EDA - torn 1

18/10/2010

## Proposta de solució al problema 1

- Doneu la definició de  $O(f)$ :

$$O(f) = \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |g(n)| \leq c|f(n)|\}$$

- El teorema mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma  $T(n) = aT(n/b) + \Theta(n^k)$  amb  $b > 1$  i  $k \geq 0$ , llavors, fent  $\alpha = \log_b a$ ,

$$T(n) = \begin{cases} \Theta(n^k) & \text{si } \alpha < k, \\ \Theta(n^k \log n) & \text{si } \alpha = k, \\ \Theta(n^\alpha) & \text{si } \alpha > k. \end{cases}$$

## Proposta de solució al problema 2

Ompliu els blancs de la forma més precisa possible.

- Quicksort ordena  $n$  elements en temps  $\Theta(n^2)$  en el cas pitjor.
- Per multiplicar dues matrius grans eficientment podem fer servir l'algorisme de Strassen.
- $2 + \cos(n) = \Theta(1)$ .
- $2\sqrt{n} + 1 + n + n^2 = \Theta(n^2)$ .
- $\log n + \log \log(n^2) = \Theta(\log n)$ .
- $\frac{n^2 - 6n}{2} + 5n = \Theta(n^2)$ .
- $n^2 - 3n - 18 = \Omega(n)$ .
- Si

$$T(n) = \begin{cases} 4 & \text{si } n = 1, \\ 3T(n/2) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(n^2)$ .

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 2T(n-1) + n^2 - 2n + 1 & \text{si } n > 1. \end{cases}$$

aleshores,  $T(n) = \Theta(2^n)$ .

**Proposta de solució al problema 3**

Sigui  $c \in \mathbb{R}$  i siguin  $f$  i  $g$  dues funcions de  $\mathbb{N}$  a  $\mathbb{R}$ , si  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$  aleshores, per definició, tenim que  $\forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : \left| \frac{f(n)}{g(n)} - c \right| \leq \epsilon$ . Fixant un  $\epsilon > 0$  arbitrari, per propietats del valor absolut si  $\left| \frac{f(n)}{g(n)} - c \right| \leq \epsilon$  aleshores  $\left| \frac{f(n)}{g(n)} \right| - |c| \leq \epsilon$  i en particular es compleix que  $|f(n)| \leq (|c| + \epsilon)|g(n)|$ . Per tant,

$$\exists c' = |c| + \epsilon > 0 : \exists m > 0 : \forall n \geq m : |f(n)| \leq c'|g(n)| \implies f \in O(g).$$

**Proposta de solució al problema 4**

Suposeu que *primer*( $n$ ) és una crida a una funció amb temps  $O(\sqrt{n})$ .

Considereu un procediment amb cos principal:

```
if (primer(n)) A;
else B;
```

Doneu fites senzilles i ajustades amb notació  $O$  per al temps d'aquest procediment, en funció de  $n$ , suposant que:

1.  $A$  triga  $O(n)$  i  $B$  triga  $O(1)$ .

$$\boxed{O(n)}$$

2.  $A$  i  $B$ , ambdòs, triguen  $O(1)$ .

$$\boxed{O(\sqrt{n})}$$

**Proposta de solució al problema 5**

```
int misteri (int n) {
 if (n == 1) return 1;
 return misteri (n-1) + 2*n - 1;
}
```

- Digueu què calcula la funció *misteri*.

$$\boxed{n^2}$$

- Doneu el seu cost en funció de  $n$ .

$$\boxed{\Theta(n)}$$

**Proposta de solució al problema 6**

Per obtenir la unió de  $A$  i  $B$  s'ha d'ordenar el vector  $B$  amb un algorisme de temps  $O(m \log m)$  en el cas pitjor (com per exemple mergesort) i cercar-hi els  $n$  elements de  $A$  amb una cerca dicotòmica. S'ha de copiar al resultat tot el conjunt  $B$  i a continuació els elements de  $A$  que no s'hagin trobat a  $B$ . El temps és  $O((n + m) \log m) = O(n \log m)$  perquè  $m \leq n$ .

Si ens demanen la intersecció, el vector  $B$  s'ordena igualment i s'han de copiar al resultat només aquells elements de  $A$  que siguin a  $B$ .

**Proposta de solució al problema 7**

L'algorisme és mergesort (ordenació per fusió) però divideix la taula d'elements a ordenar en tres parts de mida semblant i fa tres crides recursives a cada subtaula en comptes de dividir en meitats i fer dues crides recursives. El seu cost en temps,  $T(n)$ , ve donat per la recurrència:

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1, \\ 3T(n/3) + \Theta(n) & \text{si } n > 1. \end{cases}$$

i per tant  $T(n) = \Theta(n \log n)$ .

**Solució de l'Examen Parcial EDA- torn 2****18/10/2010****Proposta de solució al problema 1**

- Doneu la definició de  $\Omega(f)$ :

$$\Omega(f) = \{g : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c > 0, \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : |g(n)| \geq c|f(n)|\}$$

- El teorema mestre de resolució de recurrències substractores afirma que si tenim una recurrència de la forma  $T(n) = aT(n - c) + \Theta(n^k)$  amb  $c > 0$  i  $k \geq 0$ , llavors,

$$T(n) = \begin{cases} \Theta(n^k) & \text{si } a < 1, \\ \Theta(n^{k+1}) & \text{si } a = 1, \\ \Theta(a^{\frac{n}{c}}) & \text{si } a > 1. \end{cases}$$

**Proposta de solució al problema 2**

Ompliu els blancs de la forma més precisa possible.

- Per multiplicar dos naturals molts llargs eficientment podem fer servir l'algorisme de **Karatsuba**.
- Per ordenar  $n$  elements en temps  $\Theta(n \log n)$  en el cas pitjor podem servir l'algorisme de **mergesort (ordenació per fusió)**.
- Multipliqueu  $\log n + 6 + O(1/n)$  per  $n + O(\sqrt{n})$ . El resultat simplificat tan com és possible és  $O(\boxed{n \log n})$ .
- $\sin(n) + 10 + n = \Theta(\boxed{n})$ .
- $\frac{1}{3}n^2 + 3n \log n + 5n^8 = \Theta(\boxed{n^8})$ .
- $n\sqrt{n} + n^2 = \Theta(\boxed{n^2})$ .
- $3n \log n = \boxed{O}(n^2)$ .
- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 3T(n/2) + n - 2 & \text{si } n > 1. \end{cases}$$

$$\text{aleshores, } T(n) = \Theta(\boxed{n^{\log_2(3)}}).$$

- Si

$$T(n) = \begin{cases} 1 & \text{si } n = 1, \\ 4T(n-1) + n & \text{si } n > 1. \end{cases}$$

$$\text{aleshores, } T(n) = \Theta(\boxed{4^n}).$$

**Proposta de solució al problema 3**

Sigui  $c \in \mathbb{R}$  i siguin  $f$  i  $g$  dues funcions de  $\mathbb{N}$  a  $\mathbb{R}$ , si  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c > 0$  aleshores,  $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \frac{1}{c} > 0$  i per definició, tenim que  $\forall \epsilon > 0 : \exists m > 0 : \forall n \geq m : \left| \frac{g(n)}{f(n)} - \frac{1}{c} \right| \leq \epsilon$ . Fixant un  $\epsilon > 0$  arbitrari, per propietats del valor absolut si  $\left| \frac{g(n)}{f(n)} - \frac{1}{c} \right| \leq \epsilon$  aleshores  $\left| \left| \frac{g(n)}{f(n)} \right| - \left| \frac{1}{c} \right| \right| \leq \epsilon$  i en particular es compleix que  $|f(n)| \geq \frac{c}{c\epsilon+1}|g(n)|$ . Per tant,

$$\exists c' = \frac{c}{c\epsilon+1} > 0 : \exists m > 0 : \forall n \geq m : |f(n)| \geq c'|g(n)| \implies f \in \Omega(g).$$

**Proposta de solució al problema 4**

```

int misteri (int m, int n) {
 int result = 0;
 while (m > 0) {
 if (m % 2 != 0) result += n;
 m /= 2; n *= 2;
 }
 return result ;
}

```

- Diguen què calcula la funció *misteri*.

És l'algorisme de la multiplicació russa i calcula  $m \times n$

- Doneu el seu cost en funció de  $m$ .

$\Theta(\log m)$

**Proposta de solució al problema 5**

Considereu un procediment amb cos principal:

```

int k = f(n);
int s = 0;
for (int i = 1; i ≤ k; ++i) s += i;

```

amb  $f(n)$  una crida a la funció  $f$ .

Doneu fites senzilles i ajustades amb notació  $O$  per al temps d'aquest procediment, en funció de  $n$ , suposant que:

1. El temps de  $f(n)$  és  $O(n)$  i el valor de  $f(n)$  és  $n!$ .

$O(n!)$

2. El temps de  $f(n)$  és  $O(n)$  i el valor de  $f(n)$  és  $n$ .

$O(n)$

3. El temps de  $f(n)$  és  $O(n^2)$  i el valor de  $f(n)$  és  $n!$ .

$O(n!)$



4. El temps de  $f(n)$  és  $O(1)$  i el valor de  $f(n)$  és 0.

$$\boxed{O(1)}$$

#### Proposta de solució al problema 6

1. Amb un vector ordenat, si un element apareix repetit més de  $n/2$  vegades aquest element s'ha de trobar a la posició del mig. En temps  $O(\log n)$  (amb dues cerques dicotòmiques) es poden trobar la posició de la primera aparició d'aquest element a la taula i la posició de la darrera. Fent la resta de la segona menys la primera trobem el nombre de vegades que es repeteix aquest element i podem determinar si això passa més de  $n/2$  vegades.
2. Si el vector és de 3 elements (qualsevol constant pot servir) es calcula directament provant totes les possibilitats si hi ha un element repetit dues vegades o més. Si  $n > 3$  aleshores es divideix el vector  $A$  per la meitat en dos subvectors  $A_1$  i  $A_2$  de (quasi) la mateixa mida i es crida l'algorisme recursivament en  $A_1$  i  $A_2$ . Si  $A_1$  conté un element majoritari (com demana l'enunciat) es compara aquest element amb tots els de  $A_2$ , el que dona el nombre d'aparicions d'aquest element a  $A$ . Es segueix el mateix procediment amb  $A_2$ . Si en algun dels dos casos obtenim un element repetit més de  $n/2$  vegades, retornem aquest element i el nombre de vegades que es repeteix. En cas contrari no hi ha elements que compleixin amb la propietat.

#### Proposta de solució al problema 7

L'algorisme amagat és quicksort (ordenació ràpida) i quan tots els elements són iguals per cada increment de l'índex  $j$  hi ha un increment de l'índex  $i$  i es fa un intercanvi de cada element del vector amb sí mateix, al final de cada crida recursiva ambdòs índexos ténen per valor  $d$ , i es fan dues crides recursives a problemes de mida  $n - 1$  i  $1$  respectivament. Per tant, el cost en temps,  $T(n)$ , de l'algorisme anterior ve donat per la recurrència:

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1, \\ T(n-1) + \Theta(n) & \text{si } n > 1. \end{cases}$$

i per tant  $T(n) = \Theta(n^2)$ .

## Solució de l'Examen Parcial EDA

21/03/2011

## Proposta de solució al problema 1

Ompliu els blancs de la forma més precisa possible.

- Insertion sort ordena  $n$  elements en temps  $\Theta(n)$  en el cas millor.
- Trobar la mediana de  $n$  elements té una complexitat en temps  $\Omega(\sqrt{n})$ .
- Com de ràpid es poden multiplicar una matriu  $kn \times n$  per una  $n \times kn$  fent servir l'algorisme de Strassen?  $\Theta(k^2 n^{\log_2(7)})$ .
- Sigui  $F(n)$  el número de línies que imprimeix el programa anterior amb entrada  $n$ , aleshores,

$$F(n) = \begin{cases} \Theta(1) & \text{si } n \leq 1, \\ 2F(n/2) + \Theta(1) & \text{si } n > 1. \end{cases}$$

i per tant  $F(n) = \Theta(n)$ .

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n < 3, \\ 9T(n/3) + n^2 & \text{si } n \geq 3. \end{cases}$$

Aleshores,  $T(n) = \Theta(n^2 \log(n))$ .

- Sigui

$$T(n) = \begin{cases} 3 & \text{si } n = 1, \\ 3T(n-1) + n^3 & \text{si } n > 1. \end{cases}$$

Aleshores,  $T(n) = \Theta(3^n)$ .

- Siguin  $a = b = c = d = 01$ ,  $u = (a + b)(c + d) = 10 \times 10$ ,  $v = ac = 01 \times 01 = 1$  i  $w = bd = 01 \times 01 = 1$ . Per calcular el valor de  $u$  fem servir el mateix procediment amb  $a' = c' = 1$ ,  $b' = d' = 0$ ,  $u' = (1 + 0) \times (1 + 0) = 1$ ,  $v' = 1 \times 1 = 1$ , i  $w' = 0 \times 0 = 0$ ,  $u' - v' - w' = 0$ , aleshores  $u = v' \times 100 = 100$  i  $u - v - w = 10$ . Per tant,  $101 \times 101 = 1 \times 10000 + 10 \times 100 + 1 = 11001$ .

|   |   |   |    |   |    |    |    |                |
|---|---|---|----|---|----|----|----|----------------|
| 2 | 3 | 5 | 10 | 1 | 6  | 7  | 13 | mergesort      |
| 2 | 3 | 5 | 10 | 7 | 13 | 1  | 6  | insertion sort |
| 1 | 2 | 5 | 3  | 7 | 6  | 13 | 10 | quicksort      |
| 1 | 2 | 3 | 5  | 6 | 13 | 10 | 7  | selection sort |

| A           | B                | O  | $\Omega$ | $\Theta$ |
|-------------|------------------|----|----------|----------|
| $\log^k(n)$ | $n^\epsilon$     | sí | no       | no       |
| $\log n$    | $\log \log(n^2)$ | no | sí       | no       |
| $n^k$       | $c^n$            | sí | no       | no       |
| $2^{n+1}$   | $2^n$            | sí | sí       | sí       |
| $2^{2n}$    | $2^n$            | no | sí       | no       |

**Proposta de solució al problema 2**

- El programa troba el  $k$ -èsim element més petit de taula  $T$ .
- El cas pitjor consisteix a no poder reduir la talla del problema a resoldre recursivament en més d'un element i fer una crida recursiva a una taula amb un únic element menys. D'aquesta manera es faran el màxim nombre possible de crides recursives. Si  $T(n)$  és el cost en cas pitjor de l'algorisme anterior aleshores,

$$T(n) = \begin{cases} \Theta(1) & \text{si } n \leq 1, \\ T(n-1) + \Theta(n) & \text{si } n > 1, \end{cases}$$

on  $\Theta(n)$  és el cost no recursiu de cridar a *misteri\_2*.

- En el cas millor per retornar el  $k$ -èsim valor més petit només calen tres crides recursives a *misteri\_1* si  $k > 1$ , i dues si  $k = 1$ . Si  $k > 1$ , aquest cas millor es produeix quan en la 1a crida resulta que  $q = k - 2$ . Llavors la 2a crida es fa amb  $k = 1$ , i si aquí la partició fa que només quedi una finestra amb un sol element, aleshores la 3a crida recursiva retorna pel cas base el valor buscat. Com que en qualsevol cas es fan 2-3 crides a *misteri\_2*, que és lineal, finalment resulta que el cost en el cas millor és  $\Theta(n)$ .
- A quicksort perquè la funció *misteri\_2* és la mateixa partició que fa servir quicksort.

**Proposta de solució al problema 3**

```
#include <iostream>
#include <vector>
#include <cassert>
```

```
using namespace std;
```

```
int sqrt(int n, int l, int u) {
 assert (l*l <= n and n <= u*u);
 if (l+1 >= u) {
 if (u*u == n) return u;
 else return l;
 }
 else {
 int m = (l+u)/2;
 int sq = m*m;
 if (sq == n) return m;
 else if (sq < n) return sqrt(n, m, u);
 else return sqrt(n, l, m);
 }
}
```

```
int main(void) {
 int n;
 cin >> n;
 cout << sqrt(n, 0, n) << endl;
}
```

El cas pitjor es produeix quan  $n$  no és un quadrat. Si  $T(n)$  és el temps que es triga en aquest cas, tenim la recurrència  $T(n) = T(n/2) + \Theta(1)$  i per tant  $T(n) = \Theta(\log n)$ .

## Solució de l'Examen Parcial EDA

20/10/2011

## Proposta de solució al problema 1

- Atès que  $\sum_{i=1}^n i = n(n+1)/2$ ,  $101 + 102 + \dots + 999 + 1000 = \sum_{i=101}^{1000} i = \sum_{i=1}^{1000} i - \sum_{i=1}^{100} i = 1000 \cdot 1001/2 - 100 \cdot 101/2 = 500500 - 5050 = 495450$ .
- Establir un "com a mínim" fent servir la notació asimptòtica  $O$  no té sentit, perquè  $O$  no expressa fites inferiors, només superiors. En particular,  $O(n^2)$  inclou funcions que creixen tan lentament com vulguem.
- Ordenades de manera creixent:  $5\log(n+100)^{10}, (\ln(n))^2, \sqrt{n}, 0.001n^4 + 3n^3 + 1, 3^n, 2^{2n}$
- Siguin  $T_A, T_B, T_C$  els costos en el cas pitjor dels algorismes A, B, C.
  - $T_A(n) = 5T_A(n/2) + \Theta(n) = \Theta(n^{\log_2 5})$
  - $T_B(n) = 2T_B(n-1) + \Theta(1) = \Theta(2^n)$
  - $T_C(n) = 9T_A(n/3) + \Theta(n^2) = \Theta(n^2 \log n)$

L'algorisme C és el més eficient.

- El cas millor es dona quan  $x$  és un nombre parell. En aquest cas no hi ha carry i la suma es fa en temps  $\Theta(1)$ .

El cas pitjor es dona quan  $x$  és  $2^{n-1} - 1$ . En aquest cas el  $(n-1)$ -èsim bit de  $x$  és 0, i la resta són 1. De forma que quan es suma 1, el carry es propaga per tota la cadena de bits fins a posar el  $(n-1)$ -èsim bit a 1. Per tant, el cost és  $\Theta(n)$ .

## Proposta de solució al problema 2

- Atès que es recorren senceres les dues taules fent operacions de cost constant, el cost de `fusio2` és  $\Theta(na + nb)$ .
- A la  $i$ -èsima fusió, estem fusionant un vector de mida  $in$  amb un de mida  $n$ . Seguint l'apartat anterior, això té cost  $\Theta((i+1)n)$ . Així doncs, el cost d'aquest algorisme és  $\sum_{i=1}^{k-1} \Theta((i+1)n) = \sum_{i=2}^k \Theta(in) = \Theta(n \cdot \sum_{i=2}^k i) = \Theta(nk^2)$ .
- La funció seria:

```
vector<int> fusiok(const vector< vector<int> >& t, int e, int d) {
 if (e == d) return t[e];
 else {
 int m = (e + d)/2;
 return fusio2(fusiok(t, e, m), fusiok(t, m+1, d));
 }
}

vector<int> fusiok(const vector< vector<int> >& t) {
 return fusiok(t, 0, t.size()-1);
}
```

El cost segueix la recurrència  $T(k, n) = 2T(k/2, n) + \Theta(nk)$ , que té solució  $T(k, n) = \Theta(nk \log k)$ .

**Proposta de solució al problema 3**

Demostrem la identitat per inducció en  $n \geq 1$ . El cas base  $n = 1$  és cert ja que  $f_2 = 1$ ,  $f_1 = 1$  i  $f_0 = 0$ . Suposem ara que la identitat és certa per  $n \geq 1$  i la demostrarem per  $n + 1$ . Tenim:

$$\begin{aligned} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n+1} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} f_{n+1} + f_n & f_{n+1} \\ f_n + f_{n-1} & f_n \end{pmatrix} \\ &= \begin{pmatrix} f_{n+2} & f_{n+1} \\ f_{n+1} & f_n \end{pmatrix}. \end{aligned}$$

Per dissenyar l'algorisme de cost  $\Theta(\log n)$  farem servir la recurrència següent per calcular potències d'una matriu  $M$ :

$$M^n = \begin{cases} I & \text{si } n = 0 \\ (M^{\lfloor n/2 \rfloor})^2 & \text{si } n > 0 \text{ i és parell} \\ (M^{\lfloor n/2 \rfloor})^2 \cdot M & \text{si } n > 0 \text{ i és senar} \end{cases}$$

Aplicarem la recurrència a la matriu

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

i retornarem  $M^n[0][1]$ . Donat que  $M$  és una matriu  $2 \times 2$ , el cost de calcular aquesta recurrència és  $T(n) = T(n/2) + \Theta(1)$ , i per tant  $T(n) = \Theta(\log n)$  pel Teorema Mestre.

L'algorisme escrit en C++ és el següent:

```
void pow(int n, vector<vector<int>> &R) {
 if (n == 0) {
 R[0][0] = R[1][1] = 1;
 R[0][1] = R[1][0] = 0;
 }
 else {
 pow(n/2, R);
 vector<vector<int>> S(2, vector<int>(2));
 S[0][0] = R[0][0]*R[0][0] + R[0][1]*R[1][0];
 S[0][1] = R[0][0]*R[0][1] + R[0][1]*R[1][1];
 S[1][0] = R[1][0]*R[0][0] + R[1][1]*R[1][0];
 S[1][1] = R[1][0]*R[0][1] + R[1][1]*R[1][1];
 if (n % 2 == 0) R = S;
 else {
 R[0][0] = S[0][0] + S[0][1];
 R[0][1] = S[0][0];
 R[1][0] = S[1][0] + S[1][1];
 R[1][1] = S[1][0];
 }
 }
}
```

```
int fib (int n) {
 if (n == 0) return 0;
 vector<vector<int> > R(2, vector<int>(2));
 pow(n, R);
 return R[0][1];
}
```

## Solució de l'Examen Parcial EDA

19/3/2012

## Proposta de solució al problema 1

•

| $f(n)$          | $g(n)$           | $f = O(g)$ | $f = \Omega(g)$ | $f = \Theta(g)$ |
|-----------------|------------------|------------|-----------------|-----------------|
| $\sqrt{n}$      | $n^{2/3}$        | cert       | fals            | fals            |
| $\log(2n)$      | $\log(3n)$       | cert       | cert            | cert            |
| $n^{0.1}$       | $(\log n)^{10}$  | fals       | cert            | fals            |
| $2^n$           | $2^{n+1}$        | cert       | cert            | cert            |
| $100n + \log n$ | $n + (\log n)^2$ | cert       | cert            | cert            |

- S'ordena el vector amb un algorisme d'ordenació de cost  $O(n \log n)$  en el cas pitjor (com per exemple merge sort). Després de l'ordenació els elements repetits es troben a posicions consecutives del vector, per tant amb un recorregut lineal es poden treure les repeticions.

•

```

string pwr2bin(int n) {
 if (n == 1) return "1010"; // 10 en binari (8 + 2)
 string z = pwr2bin(n/2);
 return Karatsuba(z,z);
}

```

Sigui  $T(n)$  el cost de l'algorisme anterior amb entrada  $n$ . Aleshores  $T(n)$  satisfà la recurrència:

$$T(n) = T(n/2) + O(n^{\log_2 3})$$

perquè es fa una crida recursiva a un problema de la meitat de la mida més el cost  $O(n^{\log_2 3})$  de fer servir l'algorisme de Karatsuba amb enters de mida  $n$  a cada crida recursiva. Com que  $\log_2 3 > 0$ , domina el cost de fer servir l'algorisme de Karatsuba i per tant  $T(n) = O(n^{\log_2 3})$ .

- No. Si tenim un algorisme per multiplicar dos enters de  $n$  bits, aleshores podem elevar al quadrat un enter  $x$  de  $n$  bits amb la mateixa complexitat asimptòtica usant l'algorisme per multiplicar  $x$  per  $x$ .

## Proposta de solució al problema 2

- El nombre màxim  $C(n)$  de crides a  $f$  que pot haver-hi en un moment donat a la pila de recursió segueix la recurrència:

$$C(n) = C(n/2) + \Theta(1).$$

Per tant  $C(n) = \Theta(\log n)$ , i l'opció correcta és la c).

- El nombre  $C'(n)$  de crides a  $f$  en el cas pitjor (el qual es dona, per exemple, si  $n = 2^p + 1$  per a un cert  $p \geq 0$ ) segueix la recurrència:

$$C'(n) = 2C'(n/2) + \Theta(1).$$

Per tant  $C'(n) = \Theta(n)$ , i l'opció correcta és la b).

**Proposta de solució al problema 3**

La funció *misteri* ordena els elements de  $V[e..d]$ . Es pot raonar per inducció. Si  $e \geq d$  la funció no fa res, ja que  $V[e..d]$  ja està ordenat. Si  $e < d$ , llavors la crida *misteri*( $e, d - 1, V$ ) ordena  $V[e..d - 1]$ . Els següents bucle i intercanvi col·loquen el valor  $V[d]$  a la posició  $i$  que li correspon. Ara  $V[e..i]$  està ordenat, i tots els elements de  $V[i + 1..d]$  són més grans o iguals que els de  $V[e..i]$ . La crida *misteri*( $i + 1, d, V$ ) acaba d'ordenar finalment  $V[e..d]$ .

El cas pitjor es dona quan, després de barrejar, a la posició  $d$  hi queda l'element més petit. Si  $n = d - e + 1$ , llavors tant la primera crida com la segona a *misteri* es fan sobre vectors de mida  $n - 1$ . El cost  $T(n)$  ve determinat doncs per:

$$T(n) = 2T(n - 1) + \Theta(n)$$

i per tant  $T(n) = \Theta(2^n)$ .

Simètricament, el cas millor es dona quan, després de barrejar, a la posició  $d$  hi queda l'element més gran. Aleshores la primera crida a *misteri* es fa sobre un vector de mida  $n - 1$ , i la segona sobre un vector buit, que té cost constant. Així, el cost  $T(n)$  ve determinat per:

$$T(n) = T(n - 1) + \Theta(n)$$

i per tant  $T(n) = \Theta(n^2)$ .

**Proposta de solució al problema 4**

- Sí. Com que  $f \in O(g)$ , hi ha  $k > 0$  i  $n_1 \in \mathbb{N}$  tals que  $n \geq n_1$  implica  $f(n) \leq kg(n)$ . Com que  $\lim_{n \rightarrow \infty} g(n) = \infty$ , hi ha  $n_2 \in \mathbb{N}$  tal que si  $n \geq n_2$  llavors  $g(n) \geq k$ , i per tant  $\log g(n) \geq \log k$ . Si  $n \geq \max(n_1, n_2)$ , prenent logaritmes  $\log f(n) \leq \log k + \log g(n) \leq 2 \log g(n)$ . Així doncs,  $\log f \in O(\log g)$ .
- No. Considerem  $f(n) = 2n + 2$  i  $g(n) = n + 1$ . És clar que  $\lim_{n \rightarrow \infty} g(n) = +\infty$ . A més,  $f \in O(g)$ , però  $2^f \notin O(2^g)$  ja que  $2^{f(n)} = 2^{2n+2} = (2^{n+1})^2 = (2^{g(n)})^2$ .

**Proposta de solució al problema 5**

Descripció de l'algorisme:

```

int max_rec(const seq& A, const seq& B, int l, int u) {
 // Pre: l < u and Bl ⊆ A and Bu ⊈ A
 // Post: max{ i | l ≤ i < u and Bi ⊆ A }
 if (l == u - 1) return l;
 else {
 int h = (l + u) / 2;
 if (is_subseq(Bh, A)) return max_rec(A, B, h, u);
 else return max_rec(A, B, l, h);
 }
}

int max(const seq A&, const seq& B) {
 int n = A.length();
 int m = B.length();
 return max_rec(A, B, 0, n / m + 1);
}

```



Justificació de la correctesa:

Vegem que la funció *max\_rec* és correcta per inducció. Si  $l = u - 1$ , llavors de la precondició es dedueix que el valor buscat és  $l$ . Si  $l < u - 1$  llavors  $l < h < u$ . A més, si  $B^h \subseteq A$  llavors *max\_rec*( $A, B, h, u$ ) satisfà la precondició, i per hipòtesi d'inducció el valor retornat és el buscat. Si en canvi  $B^h \not\subseteq A$  llavors *max\_rec*( $A, B, l, h$ ) satisfà la precondició, i per hipòtesi d'inducció el valor retornat és el buscat.

Finalment, *max\_rec*( $A, B, 0, n/m + 1$ ) satisfà la precondició, ja que  $B^0 = \lambda \subseteq A$ , i raonant sobre les longituds sabem que  $B^{n/m+1} \not\subseteq A$ . Com que *max\_rec* és correcta, el valor retornat és el buscat.

Justificació del cost:

Calculem primer el cost de la part no recursiva d'una crida a *max\_rec*. El cost de construir  $B^h$  és  $O(hm)$ , i com que  $B^h$  té longitud  $hm$ , el cost de *is\_subseq*( $B^h, A$ ) és  $O(hm + n)$ . Com que  $h \leq n/m$ , el cost és  $O(n)$ .

Calculem ara el nombre de crides que es fan a *max\_rec*. Sigui  $p = u - l$ . Llavors el nombre de crides  $C(p)$  ve determinat per la recurrència

$$C(p) = C(p/2) + 1.$$

Per tant, el nombre de crides és  $C(p) = O(\log p)$ .

En conclusió el cost de l'algorisme és  $O(n \log p) = O(n \log(1 + n/m))$ .

**Solució de l'Examen Parcial EDA****22/10/2012****Proposta de solució al problema 1**

- – Cert.
- Cert.
- Fals.
- Fals.
- $T(n) = \Theta(\sqrt{n} \log(n))$ .

**Proposta de solució al problema 2**

En primer lloc observem que

$$\begin{aligned} n! &= 1 \cdot 2 \cdots n \leq n^n \\ n! &= 1 \cdot 2 \cdots n \geq (n/2)^{(n/2)}. \end{aligned}$$

En segon lloc, utilitzant el fet que  $\log_2(x)$  és creixent tenim que

$$\log_2(n!) \leq \log_2(n^n) \leq n \log_2(n) = O(n \log(n))$$

$$\log_2(n!) \geq \log_2((n/2)^{(n/2)}) \geq (n/2) \log_2(n/2) \geq (n/2) \log_2(n) - (n/2) = \Omega(n \log(n)).$$

Per tant  $\log_2(n!)$  és alhora  $O(n \log(n))$  i  $\Omega(n \log(n))$  i per tant és  $\Theta(n \log(n))^1$ .

**Proposta de solució al problema 3**

- La iteració més interior suma  $n - j$  a  $r$ . Per tant, la iteració d'enmig suma

$$\sum_{j=i+1}^n (n-j) = n(n-i) - \left( \sum_{j=1}^n j - \sum_{j=1}^i j \right) = n(n-i) - \frac{n(n+1)}{2} + \frac{i(i+1)}{2}$$

a  $r$ . Aleshores, la iteració més exterior suma

$$\begin{aligned} \sum_{i=1}^n \sum_{j=i+1}^n (n-j) &= n^3 - n \sum_{i=1}^n i - \frac{n^2(n+1)}{2} + \frac{1}{2} \sum_{i=1}^n i^2 + \frac{1}{2} \sum_{i=1}^n i \\ &= n^3 - \frac{n \cdot n(n+1)}{2} - \frac{n^2(n+1)}{2} + \frac{n(n+1)(2n+1)}{6 \cdot 2} + \frac{n(n+1)}{2 \cdot 2} \\ &= \frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} \end{aligned}$$

a  $r$ , que inicialment val 0. El grau del polinomi és doncs 3 i els coeficients (de menor a major grau) són  $0, 1/3, -1/2, 1/6$ .

Una solució alternativa consisteix en observar que la funció compta tots els possibles subconjunts de 3 elements que es poden formar en un conjunt de  $n$  elements. Per tant, la funció calcula  $\binom{n}{3} = \frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3}$ .

- El cos de la iteració més interior s'executa exactament  $n(n^2 - 3n + 2)/6$  vegades. Per tant, la funció té cost  $\Theta(n^3)$ .

---

<sup>1</sup>La inducció no va bé per demostrar fites en notació asimptòtica. Per exemple, es pot "demostrar per inducció en  $n$ " que  $n = \Theta(1)$  la qual cosa és òbviament absurda. "Demostració: Cas base ( $n = 1$ ):  $1 = \Theta(1)$ . Cas inductiu (de  $n$  a  $n + 1$ ):  $n + 1 = \Theta(1) + 1 = \Theta(1)$ ." Penseu què hi ha d'incorrecte en aquesta demostració.

**Proposta de solució al problema 4**

La funció *misteri\_a* és l'algorisme d'exponenciació ràpida, que calcula  $x^n$  amb cost  $\Theta(\log(n))$  (el mateix en els casos pitjor, mitjà i millor). Per tant, *misteri\_b* eleva a la potència  $n$  cada element del vector  $V$ , és a dir, calcula  $V^n$ , amb cost  $\Theta(m \log(n))$  (el mateix en els casos pitjor, mitjà i millor).

**Proposta de solució al problema 5**

S'aplica un algorisme d'exponenciació ràpida on els productes es fan amb l'algorisme de Karatsuba.

Sigui  $k = \log_2(3)$ , l'exponent de l'algorisme de Karatsuba. Llavors (ignorant els productes dels senars que, com a molt són el doble):

1. Es multiplica  $m \times m$  per obtenir  $m^2$ . És un Karatsuba de  $n$  bits per  $n$  bits, per tant cost  $n^k$ .
2. Es multiplica  $m^2 \times m^2$  per obtenir  $m^4$ . És un Karatsuba de  $2n$  bits per  $2n$  bits, per tant cost  $(2n)^k$ .
3. Es multiplica  $m^4 \times m^4$  per obtenir  $m^8$ . És un Karatsuba de  $4n$  bits per  $4n$  bits, per tant cost  $(4n)^k$ .
- ...
- log<sub>2</sub>(m/2).** Es multiplica  $m^{(m/2)} \times m^{(m/2)}$  per obtenir  $m^m$ . És un Karatsuba de  $(m/2)n$  bits per  $(m/2)n$  bits, per tant cost  $((m/2)n)^k$ .

En total,

$$T(n) = \sum_{i=0}^{n-1} (2^i n)^k = n^k \sum_{i=0}^{n-1} 2^{ik} = \Theta(n^k 2^{kn})$$

on s'utilitza el fet que  $\sum_{i=0}^n 2^{ik} = \Theta(2^{kn})$ .

Alternativament, es podria fer la majoració següent:

$$T(n) \leq n((m/2)n)^k = O(n^{k+1} 2^{kn}).$$

**Proposta de solució al problema 6**

```
double zero(double a, double b, double tol) {
 double c = (a+b)/2;
 if (b - a ≤ tol or f(c) == 0) return c;
 else if (f(c) * f(b) < 0) return zero(c, b, tol);
 else return zero(a, c, tol);
}
```

En el cas pitjor a cada crida recursiva la longitud de l'interval es redueix en un factor  $\frac{1}{2}$ , i després de  $k$  crides l'interval té doncs longitud  $\frac{b-a}{2^k}$ . El programa s'atura quan aquesta longitud és més petita o igual que  $tol$ . Per tant, el nombre de crides recursives és com a molt  $O(\log(\frac{b-a}{tol}))$ . Com que a cada crida recursiva el treball que es fa és  $O(1)$ , el programa costa en el cas pitjor  $O(\log(\frac{b-a}{tol}))$ .

**Solució de l'Examen Parcial EDA****18/3/2013****Proposta de solució al problema 1**

A:  $T(n) = 5T(n/2) + \Theta(n)$ . És a dir  $T(n) = \Theta(n^{\log_2 5})$ .

B:  $T(n) = 2T(n-1) + \Theta(1)$ . És a dir  $T(n) = \Theta(2^n)$ .

C:  $T(n) = 9T(n/3) + \Theta(n^2)$ . És a dir  $T(n) = \Theta(n^2 \log n)$ .

D: El B és exponencial, i els altres com a molt polinòmics; per tant el descartem. Entre A i C ens quedem amb C perquè  $\log_2 5 > 2.01$  i  $\lim_{n \rightarrow \infty} n^{2.01} / n^2 \log n = \lim_{n \rightarrow \infty} n^{0.01} / \log n = \infty$ .

**Proposta de solució al problema 2**

a) Cert. Justificació: El nombre de compostos a l'interval  $[1, \dots, n]$  és  $n - \pi(n)$ . Per tant, és suficient comprovar que  $n - \pi(n) > \pi(n)$  per  $n$  suficientment gran. Fem-ho. Del fet que  $\pi(n) = \Theta(n / \log n)$  podem concloure que

$$(n - \pi(n)) / \pi(n) = n / \pi(n) - 1 = \Theta(\log n).$$

Donat que  $\lim_{n \rightarrow \infty} \log n = \infty$  en deduïm que  $n - \pi(n) > \pi(n)$  per  $n$  suficientment gran.

b) Fals. Justificació: El nombre de quadrats a l'interval  $[1, \dots, n]$  és com a molt  $\sqrt{n}$ . Per tant, és suficient comprovar que  $\sqrt{n} < \pi(n)$  per  $n$  suficientment gran. Fem-ho. Del fet que  $\pi(n) = \Theta(n / \log n)$  podem concloure que

$$\pi(n) / \sqrt{n} = \Theta(\sqrt{n} / \log n).$$

Donat que  $\lim_{n \rightarrow \infty} \sqrt{n} / \log n = \infty$  en deduïm que  $\pi(n) > \sqrt{n}$  per  $n$  suficientment gran.

**Proposta de solució al problema 3**

a) Això són tres bucles imbricats, cadascun d' $n$  iteracions, i amb cost  $\Theta(1)$  per cada iteració. Total:  $\Theta(n^3)$ .

b) Això tornen a ser tres bucles imbricats, però ara el cost és

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=j}^{n-1} \Theta(1) \leq \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} \Theta(1) = \Theta(n^3).$$

Per altra banda tenim que

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=j}^{n-1} \Theta(1) \geq \sum_{i=0}^{n/3} \sum_{j=n/3}^{2n/3} \sum_{k=2n/3}^{n-1} \Theta(1) = \Theta((n/3)^3) = \Theta(n^3).$$

Total: el cost és alhora  $O(n^3)$  i  $\Omega(n^3)$  i per tant  $\Theta(n^3)$ .

c) Construir  $Q$  té cost  $\Theta(n^2)$ . Ordenar  $Q$  té cost  $\Theta(n^2 \log n^2) = \Theta(n^2 \log n)$ . L'última fase té cost  $\Theta(n \log n^2) = \Theta(n \log n)$  perquè per a cada  $k$  busquem  $s - V[k]$  a  $Q$  fent una cerca dicotòmica. Total:  $\Theta(n^2 \log n)$ .

d) Construir  $Q$  té cost  $\Theta(n^2)$ . Ordenar  $V$  té cost  $\Theta(n \log n)$ . L'última fase té cost  $\Theta(n^2 \log n)$  perquè per a cada  $\ell$  busquem  $s - Q[\ell]$  a  $V$  fent una cerca dicotòmica. Total:  $\Theta(n^2 \log n)$ .

**Proposta de solució al problema 4**

1. El següent codi satisfà els requeriments:

```

int g1(int n) {
 if (n ≤ 2) return 1;
 int v1, v2, v3;
 v1 = v2 = v3 = 1;
 for (int i = 2; i < n; ++i) {
 int aux = v2 + v3;
 v3 = v2;
 v2 = v1;
 v1 = aux;
 }
 return v1;
}

```

Per a  $n \geq 3$  es fan  $n - 2$  iteracions, cadascuna de les quals amb cost temporal  $\Theta(1)$ . D'aquí el cost en temps  $\Theta(n)$ .

2. Aquí és el codi amb les caixetes plenes:

```

void misteri(const matrix<int>& m, int k, matrix<int>& p) {
 if (k == 0)
 for (int i = 0; i < 3; ++i) p[i][i] = 1;
 else {
 misteri(m, k/2, p);
 p = p * p;
 if (k % 2 == 1) p * m;
 }
}

```

**Proposta de solució al problema 5**

Primer busquem la posició  $p \geq 1$  d'algun  $\infty$  així:

```

int p = 1;
while (V[p] ≠ ∞) p *= 2;

```

Comencem en 1 perquè ens diuen que  $n \geq 1$ . El nombre d'iteracions serà el mínim  $i \geq 0$  tal que  $2^i \geq n$ . És a dir  $i = \Theta(\log n)$ . Un cop tenim aquest  $p$  fem una cerca dicotòmica entre  $v[p/2]$  i  $v[p]$  buscant la posició del primer  $\infty$  així:

```

int l = p/2;
int r = p;
while (l < r - 1) {
 int q = (l + r)/2;
 if (V[q] == ∞) r = q;
 else l = q;
}
return r;

```

El nombre d'iteracions d'aquesta segona fase és  $\Theta(\log(p/2))$ . Donat que  $p/2 < n \leq p$ , tenim  $p/2 = \Theta(n)$ . Per tant això són  $\Theta(\log n)$  iteracions. Cost total:  $\Theta(\log n) + \Theta(\log n) = \Theta(\log n)$ .

**Solució de l'Examen Parcial EDA****21/10/2013****Proposta de solució al problema 1**

- $T(n) = \Theta(n)$ .
- Les tres:  $O(n^2)$ ,  $\Theta(n \log n)$  i  $\Omega(n \log n)$ .
- $f(n) = n \log \log n$ .
- $T(n) = \Theta(n^2)$ .
- $T(n) = 3T(n/2) + \Theta(n)$ .
- $T(n) = \Theta(n)$ .
- $T(n) = \Theta(1)$ .
- $T(n) = \Theta(n \log n)$ .

**Proposta de solució al problema 2**

a) El primer fa  $n - 1$  iteracions de cost constant i, per tant, té un cost  $\Theta(n)$ . El cost del segon es pot descriure amb la recurrència  $T(n) = 2T(n/2) + \Theta(1)$ , que pel teorema mestre de les recurrències divisores, és  $\Theta(n)$ .

b) El primer compara les variables *min* i *max* amb cada posició del vector  $A[i]$ , per a  $1 \leq i \leq n - 1$ , de manera que el nombre total de comparacions és  $2n - 2$ . En el cas que  $n$  és potència de dos, suposem  $n = 2^k$ , l'arbre de recursió generat pel segon algorisme serà un arbre binari complet on cada fulla correspondrà a un subvector de dos elements. Per tant, l'alçada de l'arbre serà  $k - 1$ , el nombre de fulles  $2^{k-1} = n/2$  i el nombre de nodes interns (corresponents a crides amb subvectors de mida més gran que 2)

$$\sum_{i=0}^{k-2} 2^i = 2^{k-1} - 1.$$

Tenint en compte que en el cas base (subvectors de mida 2) es fa només una comparació i que en cada crida del cas general (subvectors de mida  $> 2$ ) se'n fan 2, el total de comparacions serà

$$2^{k-1} + 2(2^{k-1} - 1) = 2^{k-1} + 2^k - 2 = 3 \cdot 2^{k-1} - 2 = 3n/2 - 2,$$

que millora les  $2n - 2$  comparacions de l'algorisme iteratiu.

**Proposta de solució al problema 3**

a) Creem un vector addicional  $P[1, \dots, n]$  que dirà, per cada  $i = 1, \dots, n$ , la posició de  $i$  a  $V$ . Per omplir  $P$  simplement recorrem l'entrada  $i$ , per cada  $i = 1, \dots, n$ , fem  $P[V[i]] = i$ . Finalment fem  $D[i] = |P[i + 1] - P[i]|$  per cada  $i = 1, \dots, n - 1$ . El cost és  $\Theta(n)$  perquè són dos recorreguts de dos vectors de longitud  $n$ .

b) Creem un vector addicional  $P[1, \dots, n]$  que dirà, per cada  $i = 1, \dots, n$ , la posició de l' $i$ -èssim element més gran de  $V$ . Per fer-ho, ordenarem el vector  $V$  fent un mergesort modificat de manera que, quan col·loqui l'element  $V[k]$  a la seva posició definitiva  $i$ , posi  $P[i] = k$ . Finalment fem  $D[i] = |P[i + 1] - P[i]|$  per cada  $i = 1, \dots, n - 1$ . El cost és  $\Theta(n \log n)$  per la fase on s'executa el mergesort modificat, i  $\Theta(n)$  per la fase on es genera  $D$ . El cost total és doncs  $\Theta(n \log n)$ .

**Proposta de solució al problema 4**

La funció *misteri\_a* és l'algorisme d'exponenciació ràpida, que calcula  $x^n$  amb cost  $\Theta(\log(n))$  (el mateix en els casos pitjor, mitjà i millor). Per tant, *misteri\_b* eleva a la potència  $n$  cada element del vector  $V$ , és a dir, calcula  $V^n$ , amb cost  $\Theta(m \log(n))$  (el mateix en els casos pitjor, mitjà i millor).

**Proposta de solució al problema 5**

Per una banda tenim que  $S(n) \geq 1$  i per l'altra que

$$S(n) = \sum_{i=1}^n \frac{1}{i^2} = 1 + \sum_{i=2}^n \frac{1}{i^2} \leq 1 + \int_1^n \frac{1}{x^2} dx = 1 + \left[ -\frac{1}{x} \right]_1^n = 2 - \frac{1}{n}.$$

Per tant  $S(n)$  és alhora  $\Omega(1)$  i  $O(1)$  i per tant  $\Theta(1)$ .

**Proposta de solució al problema 6**

L'enunciat és fals. Un contraexemple seria  $f(n) = 2^n$  i  $c = 2$ . En efecte

$$\lim_{n \rightarrow \infty} \frac{f(cn)}{f(n)} = \lim_{n \rightarrow \infty} \frac{2^{2n}}{2^n} = \lim_{n \rightarrow \infty} 2^n = \infty$$

i per tant  $f(cn)$  no és  $\Theta(f(n))$ .

**Solució de l'Examen Parcial EDA****24/3/2014****Proposta de solució al problema 1**

a)  $H(n) = H(n-1) + 1$  i  $W(n) = 2W(n-1) + 2$ .

b)  $L(n) = 2L(n-1) + 2$ .

c) El teorema mestre dóna  $H(n) = \Theta(n)$ ,  $W(n) = \Theta(2^n)$  i  $L(n) = \Theta(2^n)$ . Siguin  $A_1(n)$  i  $A_2(n)$  les àrees per un arbre complet de  $4^n$  fulles en la primera i la segona construcció, respectivament. Tenint en compte que  $4^n = 2^{2n}$  tenim que  $A_1(n) = H(2n)W(2n) = \Theta(2^{2n})\Theta(2n) = \Theta(2^{2n}n)$ . Per altra banda  $A_2(n) = L(n)^2 = \Theta(2^{2n})$ . Donat que  $\lim_{n \rightarrow \infty} 2^{2n}n/2^{2n} = +\infty$ , en deduïm que  $A_2(n)$  és  $O(A_1(n))$  però no  $\Theta(A_1(n))$ . Per tant, la segona construcció és asimptòticament més eficient que la primera.

**Proposta de solució al problema 2**

El cos de cada iteració de cada bucle és  $\Theta(1)$  i per tant només cal comptar quantes iteracions fa cada bucle. Si  $y_t$  denota el valor de  $y$  al final de la  $t$ -èssima iteració, el que busquem és el mínim  $t \geq 0$  tal que  $y_t > n$ . Determinem  $y_t$  per cada bucle:

1.  $y_t = 1 + 2 + 3 + 4 + \dots + (t+1) = \sum_{j=1}^{t+1} j = \Theta(t^2)$ .
2.  $y_t = 2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^t = \Theta(2^t)$ .
3.  $y_t = 2^0 \cdot 2^1 \cdot 2^2 \cdot 2^3 \dots 2^t = 2^{\sum_{j=1}^t j} = 2^{\Theta(t^2)}$ .
4.  $y_t = 1 \cdot 2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \cdot 2^{2^3} \dots 2^{2^{t-1}} = 2^{\sum_{j=0}^{t-1} 2^j} = 2^{\Theta(2^t)}$ .

Ara busquem el mínim  $t$  que satisfà  $y_t > n$ :

1. Volem  $\Theta(t^2) > n$  i el mínim tal  $t$  és  $t = \Theta(\sqrt{n})$ .
2. Volem  $\Theta(2^t) > n$  i el mínim tal  $t$  és  $t = \Theta(\log n)$ .
3. Volem  $2^{\Theta(t^2)} > n$ , o  $\Theta(t^2) > \log n$ , i el mínim tal  $t$  és  $t = \Theta(\sqrt{\log n})$ .
4. Volem  $2^{\Theta(2^t)} > n$ , o  $\Theta(2^t) > \log n$ , i el mínim tal  $t$  és  $t = \Theta(\log \log n)$ .

**Proposta de solució al problema 3**

a) Considerem una  $k$  arbitrària tal que  $i < k < j$ . Hi ha dos casos:

- Si  $T[i] > T[k]$ , llavors el parell  $(i, k)$  és una inversió.
- Altrament, com que  $T[k] \geq T[i] > T[j]$ , el parell  $(k, j)$  és una inversió.

En qualsevol cas, per cada  $k$  tal que  $i < k < j$  podem comptar almenys una inversió. Comptant també la inversió  $(i, j)$ , tenim almenys  $j - i$  inversions.

b) De l'apartat anterior deduïm que, si  $T$  és un vector amb com a molt  $N$  inversions, aleshores tota inversió  $(i, j)$  ha de satisfer  $j - i \leq N$ . A continuació es mostra una possible implementació que utilitza aquesta observació:

```
bool cerca(int x, const vector<int>& T, int l, int r) {
 if (r-l+1 < 2*N+1) {
 for (int i = l; i <= r; ++i) if (T[i] == x) return true;
 return false;
 }
```



```

 }
 int m = (l+r)/2;
 if (T[m] > x) {
 for (int k = 1; k ≤ N; ++k) if (T[m+k] == x) return true;
 return cerca(x, T, l, m-1);
 }
 if (T[m] < x) {
 for (int k = 1; k ≤ N; ++k) if (T[m-k] == x) return true;
 return cerca(x, T, m+1, r);
 }
 return true;
} }

bool cerca(int x, const vector<int>& T) {
 return cerca(x, T, 0, T.size() - 1);
}

```

Com que  $N$  és una constant, el cost dels bucles dels casos recursius és  $\Theta(1)$ . La recurrència que descriu el cost  $C(n)$  de *cerca* per a vectors de mida  $n$  en el cas pitjor (per exemple, quan l'enter  $x$  no pertany al vector  $T$ ) és doncs

$$C(n) = C(n/2) + \Theta(1),$$

la solució de la qual és  $C(n) = \Theta(\log n)$  d'acord amb el Teorema Mestre de Recurrències Divisores.

#### Proposta de solució al problema 4

1) Resposta:  $\Theta(n)$ . Justificació: Donat que  $-1 \leq \cos(x) \leq 1$  per a qualsevol  $x \in \mathbb{R}$ , tenim que  $\log(3) \leq \log(\cos(n\pi) + 4) \leq \log(5)$  per a qualsevol  $n \geq 0$ . Per tant  $\log(\cos(n\pi) + 4) = \Theta(1)$  i  $f(n) = n\Theta(1) = \Theta(n)$ .

2) Resposta: 1. Justificació: Si ja hem calculat  $7^{2^i}$  podem obtenir  $7^{2^{i+1}}$  simplement elevant-lo al quadrat. Fem-ho, tot mòdul 10:  $7^2 = 9$ ,  $7^4 = 9^2 = 1$ ,  $7^8 = 1^2 = 1$ . I d'aquí no ens movem perquè  $1^2 = 1$ . Per tant  $7^{1024} = 7^{2^{10}} = 1$ .

3)  $T(n) = 7T(n/2) + \Theta(n^2)$ .

4) Resposta:  $\Theta(n^2)$ . Justificació: El cost en el cas mitjà és

$$\frac{1}{2^n} \cdot \Theta(n^4) + \left(1 - \frac{1}{2^n}\right) \cdot \Theta(n^2)$$

que és  $\Theta(n^4/2^n) + \Theta(n^2) - \Theta(n^2/2^n)$  i per tant  $\Theta(n^2)$  perquè les altres dues funcions tendeixen a 0 quan  $n$  tendeix a infinit.

**Solució de l'Examen Parcial EDA****13/10/2014****Proposta de solució al problema 1**

a) Veurem que la suma és  $O(n^4)$  i  $\Omega(n^4)$ . Primer,  $\sum_{i=0}^n i^3 \leq n \cdot n^3 = n^4 = O(n^4)$ . Segon, si  $n$  és parell, llavors  $\sum_{i=0}^n i^3 \geq \frac{n}{2} \cdot \left(\frac{n}{2}\right)^3 \geq \frac{1}{16}n^4 = \Omega(n^4)$ . I si  $n$  és senar, llavors  $\sum_{i=0}^n i^3 = \sum_{i=0}^{n-1} i^3 + n^3 \geq \frac{1}{16}(n-1)^4 + n^3 = \Omega(n^4)$ . En qualsevol cas obtenim que  $\sum_{i=0}^n i^3 = \Omega(n^4)$ .

b)  $g$  creix més ràpid que  $f$ . Per justificar-ho veurem que  $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ . Sigui  $h(n) = \log_2(f(n)/g(n))$ . Simple manipulació dóna  $h(n) = \sqrt{\log n} - \log_2 n$  i per tant  $\lim_{n \rightarrow +\infty} h(n) = -\infty$ . Això implica que  $\lim_{n \rightarrow +\infty} 2^{h(n)} = 0$  i per tant  $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ . Pel criteri del límit en concloem que  $f(n) = O(g(n))$  i  $f(n) \neq \Omega(g(n))$ . Per tant  $g$  creix més ràpid que  $f$ .

**Proposta de solució al problema 2**

a)  $B$  conté els elements de  $A$  ordenats de més petit a més gran. Per justificar-ho, fixeuvos que, abans d'executar l'últim bucle,  $C[i]$  conté el nombre d'elements de  $A$  que estan entre 0 i  $i$ , ambdós inclosos, per  $0 \leq i \leq k$ . Per tant, l'últim bucle posa a  $B$  tantes còpies de cada element de  $A$  com hi ha al vector  $A$ , de dreta a esquerra i de més gran a més petit.

b) Quan  $k = O(n)$ , el cost és  $\Theta(n)$ . Per justificar-ho, analitzem línia a línia. La primera línia té cost  $\Theta(1)$ . La segona línia té cost  $\Theta(k)$ . La tercera línia té cost  $\Theta(n)$ . La quarta línia té cost  $\Theta(k)$ . La cinquena línia té cost  $\Theta(n)$ . I l'últim bucle té cost  $\Theta(n)$ . El cost global és doncs  $\Theta(n+k)$  i, quan  $k = O(n)$ , el terme  $n$  guanya i obtenim cost  $\Theta(n)$ .

c) Primer computem  $C[0, \dots, k]$  com a *misteri* de manera que  $C[i]$  és el nombre d'elements de  $A$  que estan entre 0 i  $i$ , ambdós inclosos. I després, per cada  $i = 0, \dots, m-1$ , si  $a_i > 0$  escrivim  $C[b_i] - C[a_i - 1]$ , i si  $a_i = 0$  escrivim  $C[b_i]$ . El cost per calcular  $C$  és  $\Theta(n+k)$  i el cost per tractar els  $(a_i, b_i)$  és  $\Theta(m)$ . El cost global és doncs  $\Theta(n+k+m)$ .

**Proposta de solució al problema 3**

a)  $R_{i,j}$  és la probabilitat d'anar de  $i$  a  $j$  en exactament dues tirades. Per justificar-ho, només cal interpretar  $k$  com el pas intermig i adonar-se que  $P_{i,k}P_{k,j}$  és la probabilitat que en la primera tirada anem de  $i$  a  $k$  i en la segona anem de  $k$  a  $j$ . Sumant sobre  $k = 1, \dots, 64$  obtenim el que volem.

b)

```
void probabilities (const Matrix& P, int t, Matrix& Q) {
 if (t == 0) identity (Q, 64);
 else {
 Matrix R, S;
 probabilities (P, t/2, R);
 multiply(R, R, S);
 if (t % 2 == 1) multiply(S, P, Q);
 else Q = S;
 }
}
```

```
void identity (Matrix& Q, int n) {
 Q = Matrix(n, Row(n, 0.0));
 for (int i = 0; i < n; ++i) Q[i][i] = 1.0;
```

```

}

void multiply(const Matrix& A, const Matrix& B, Matrix& C) {
 C = Matrix(A.size (), Row(B[0].size (), 0.0));
 for (int i = 0; i < A.size (); ++i) {
 for (int j = 0; j < B[0].size (); ++j) {
 for (int k = 0; k < B.size (); ++k) {
 C[i][j] += A[i][k]*B[k][j];
 }
 }
 }
}

```

Cost:  $T(t) = T(t/2) + \Theta(1)$  i per tant  $T(t) = \Theta(\log t)$  pel cas  $\alpha = k = 0$  del teorema mestre de recurrències divisores.

#### Proposta de solució al problema 4

a) Resposta:  $\Omega(2^n)$ .

Justificació: Sigui  $B$  el número d'entrades que causen cost asimptòtic  $\Theta(n^2)$ , i sigui  $G = 2^n - B$  el número d'entrades amb cost asimptòticament inferior a  $n^2$ . Llavors el cost en el cas mitjà és  $(B/2^n)\Theta(n^2) + (1 - B/2^n)f(n)$  on  $f(n)$  és una funció de creixement asimptòticament inferior a  $n^2$ . Per a què aquest cost sigui  $\Theta(n^2)$  és necessari que, per  $n$  suficientment gran,  $B$  sigui al menys una fracció de  $2^n$ , és a dir,  $B \geq c2^n$  per alguna  $c > 0$ , o  $B = \Omega(2^n)$ . La mateixa fórmula permet veure que la condició  $B = \Omega(2^n)$  també és suficient per garantir cost  $\Theta(n^2)$ .

b) Resposta:  $\frac{9}{10} \left(\frac{1}{10}\right)^k$

Justificació: Perquè l'algorisme faci **exactament**  $k$  iteracions cal que  $A$  acabi amb  $k$  9's i el dígit just anterior sigui diferent de 9. La probabilitat d'aquest esdeveniment és el què hem escrit.

c) Resposta:  $T(n) = \Theta(1)$ .

Justificació: Amb la notació de classe per les recurrències substractores tenim que  $a = 1/10$ ,  $c = 1$  i  $k = 0$ . Per tant, donat que  $a < 1$ , el resultat és  $T(n) = \Theta(n^k) = \Theta(1)$ .

d)  $T(n)$  correspon al cost asimptòtic en el cas mitjà amb la distribució de probabilitat especificada a l'apartat b).

**Solució de l'Examen Parcial EDA**  
**Proposta de solució al problema 1**

23/03/2015

a)  $\lfloor n/2 \rfloor - 1$

b) 0

c)  $\lfloor n/17 \rfloor - 1$

d) El cost de l'algorisme es pot expressar com:

$$\sum_{\substack{x=2 \\ x \text{ primer}}}^n \Theta(\lfloor n/x \rfloor - 1) + \sum_{\substack{x=2 \\ x \text{ no primer}}}^n \Theta(1).$$

El segon sumatori és  $O(n)$ . Per altra banda, com que  $\lfloor n/x \rfloor - 1$  és  $\Theta(n/x)$ , el primer sumatori és equivalent a

$$\sum_{\substack{x=2 \\ x \text{ primer}}}^n \Theta(n/x) = n \sum_{\substack{x=2 \\ x \text{ primer}}}^n \Theta(1/x) = \Theta(n \log \log n).$$

El resultat és doncs

$$\Theta(n \log \log n) + O(n) = \Theta(n \log \log n).$$

e) El cost no millora, continua essent  $\Theta(n \log \log n)$ , perquè el cost en aquest cas té una expressió similar a l'anterior amb l'única diferència que ara els sumatoris arriben només fins a  $\sqrt{n}$ . Per tant, l'expressió final que un obté és  $\Theta(n \log \log \sqrt{n})$ , que és el mateix que  $\Theta(n \log \log n)$ , ja que  $\log \log \sqrt{n} = \log(\frac{1}{2} \log n) = \log \frac{1}{2} + \log \log n = \Theta(\log \log n)$ .

**Proposta de solució al problema 2**

a) El mínim  $n$  tal que  $n^3 \geq 10n^{2.81}$ , és a dir,  $n^{0.19} \geq 10$ , és  $n = \lceil 10^{\frac{1}{0.19}} \rceil = 183299$ .

b) El mínim  $n$  tal que  $10n^{2.81} \geq 100n^{2.38}$ , és a dir,  $n^{0.43} \geq 10$ , és  $n = \lceil 10^{\frac{1}{0.43}} \rceil = 212$ .

**Proposta de solució al problema 3**

a) Una solució consisteix en ordenar els intervals en ordre creixent per l'extrem esquerre en temps  $\Theta(n \log n)$ , i després processar-los de la manera descrita a continuació. Recorrem la seqüència d'esquerra a dreta mantenint l'extrem esquerre mínim (*eem*) i l'extrem dret màxim (*edM*) vistos des de l'últim interval que hem escrit a la sortida. Si el següent interval de la seqüència té un extrem esquerre més gran que l'*edM* llavors podem *tancar* l'interval [*eem*, *edM*], afegir-lo a la sortida, i actualitzar *eem* i *edM* als extrems esquerre i dret de l'interval que acabem de processar. En cas contrari actualitzem l'*edM* si és necessari, és a dir, si l'extrem dret de l'interval processat és més gran que l'*edM*. El cost d'aquesta fase és  $\Theta(n)$  i per tant el cost total és  $\Theta(n \log n)$ .<sup>2</sup>

b) En primer lloc calculem la unió dels intervals igual que en el primer apartat, amb cost  $\Theta(n \log n)$ . Després determinem, per cada punt  $p_i$ , si està dins d'algun interval de la unió

<sup>2</sup>Una segona solució seria un algorisme de dividir-i-vèncer semblant a l'ordenació per fusió.

o no. Quan  $m$  és gran, com és el cas si  $m = n$ , una bona solució consisteix a ordenar  $p$  en ordre creixent, i després “fusionar” intervals i punts en temps lineal. A cada pas de la fusió considerem un interval  $[a_i, b_i]$  i un punt  $p_j$ . Si  $b_i < p_j$ , l'interval es descarta i avancem a la seqüència d'intervals. Si  $a_i \leq p_j \leq b_j$ , incrementem el comptador i avancem a les dues seqüències. Si  $p_j < a_i$ , el punt es descarta i avancem a la seqüència de punts. Quan no quedin punts, el procés acaba. El cost de la segona fase és  $\Theta(m \log m) + \Theta(n + m)$ . Per  $m = n$ , això és  $\Theta(n \log n)$ .

c) Farem servir un altre algorisme. Un punt pertany a la unió si i només si pertany a algun dels intervals de la seqüència d'entrada, i per tant podem determinar si pertany a la unió en temps  $\Theta(n)$  simplement recorrent la seqüència d'intervals tal com ens ve donada (sense processar-la prèviament). Donat que  $m \leq 5$ , això són no més de 5 recorreguts de cost  $\Theta(n)$  cadascun i per tant el cost total és  $\Theta(n)$ .<sup>3</sup>

#### Proposta de solució al problema 4

a) Resposta:  $\Theta(3^{\log_2(n)}) \neq \Theta(3^{\log_4(n)})$ . Justificació: Siguin  $f(n) = 3^{\log_2(n)}$  i  $g(n) = 3^{\log_4(n)} = 3^{\log_2(n)/2}$  de manera que  $f(n)/g(n) = 3^{\log_2(n) - \log_2(n)/2} = 3^{\log_2(n)/2}$ . Com que  $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$ ,  $f(n)$  creix estrictament més ràpid que  $g(n)$ .

b) Cal calcular  $2^1 \cdot \dots \cdot 2^{100} = 2^{5050} \pmod{9}$ . Com que  $2^6 = 1 \pmod{9}$  i  $5050 = 4 \pmod{6}$ , tenim  $2^{5050} = 2^4 = 7 \pmod{9}$ .

c) Ordenades d'ordre de creixement més petit a més gran, les funcions són

$$(\ln(n))^2, n^{1/3}, \sqrt{n}, n^4 - 3n^3 + 1.$$

d) Les tres recurrències són:

$$A(n) = \Theta(n) + 5A(n/2) = \Theta(n^{\log_2 5}).$$

$$B(n) = \Theta(1) + 2B(n-1) = \Theta(2^n).$$

$$C(n) = \Theta(n^2) + 9C(n/3) = \Theta(n^2 \log n).$$

C és la més eficient perquè  $\log n$  creix més lentament que  $n^{\log_2 5 - 2} = n^{0.3219\dots}$ .

---

<sup>3</sup>Una segona solució, menys eficient, seria primer calcular la unió en forma d'intervals disjunts ordenats com en el primer apartat en temps  $\Theta(n \log n)$ , i després fer  $m$  cerques dicotòmiques en temps  $\Theta(m \log n)$ . Quan  $m$  és una constant, el cost total és  $\Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

### Proposta de solució al problema 1

19/10/2015

(a) Anàlisi de cost:

- *En temps*: el cost de les inicialitzacions és  $\Theta(k)$ , per la creació del vector  $f$ . Com que el bucle s'executa  $\Theta(k)$  vegades i cada iteració costa temps  $\Theta(1)$ , la contribució al cost del bucle és  $\Theta(k)$ . En total el cost és doncs  $\Theta(k)$ .
- *En espai*:  $\Theta(k)$ , ja que el consum de memòria està dominat per la creació del vector  $f$ , de mida  $k + 1$ .

(b) Per inducció.

- *Cas base*:  $k = 2$ . Aleshores efectivament

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k-1} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_2 & f_1 \\ f_1 & f_0 \end{pmatrix} = \begin{pmatrix} f_k & f_{k-1} \\ f_{k-1} & f_{k-2} \end{pmatrix}.$$

- *Cas inductiu*:  $k > 2$ . Per hipòtesi d'inducció:

$$\begin{aligned} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{k-1} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} f_k & f_{k-1} \\ f_{k-1} & f_{k-2} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \\ &= \begin{pmatrix} f_k + f_{k-1} & f_k \\ f_{k-1} + f_{k-2} & f_{k-1} \end{pmatrix} = \begin{pmatrix} f_{k+1} & f_k \\ f_k & f_{k-1} \end{pmatrix}. \end{aligned}$$

(c) Una possible solució:

```
typedef vector<vector<int>> matrix;
```

```
matrix mult(const matrix& A, const matrix& B) {
 assert (A.size () == A[0].size ());
 assert (B.size () == B[0].size ());
 assert (A.size () == B.size ());
 int n = A.size ();
 matrix C(n, vector<int>(n, 0));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < n; ++j)
 for (int k = 0; k < n; ++k)
 C[i][j] += A[i][k] * B[k][j];
 return C;
}
```

```
matrix misteri (const matrix& M, int q) {
 int s = M.size ();
 if (q == 0) {
 matrix R(s, vector<int>(s, 0));
 for (int i = 0; i < s; ++i) R[i][i] = 1;
 return R;
 }
 else {
```

```

matrix P = misteri (M, q/2);
if (q % 2 == 0) return mult(P, P);
else return mult(mult(P, P), M);
} }

```

```

int fib2 (int k) {
 if (k ≤ 1) return k;
 matrix M = { {1, 1}, {1, 0} };
 matrix P = misteri (M, k-1);
 return P[0][0];
}

```

- (d) Primer analitzem el cost de  $\text{misteri}(M, k)$  en funció de  $k$ , que anomenarem  $C(k)$ . Observem que les crides a  $\text{mult}$  sempre es fan amb matrius  $2 \times 2$ , i per tant triguen temps constant. Per tant el cost no recursiu és constant i tenim que  $C(k) = C(k/2) + \Theta(1)$ , d'on aplicant el Teorema Mestre de Recurrències Divisores s'obté que  $C(k) = \Theta(\log k)$ .

Així doncs, el cost de  $\text{fib2}(k)$  és  $C(k-1) + \Theta(1) = \Theta(\log k)$ .

### Proposta de solució al problema 2

- (a) Un cas millor es dona quan  $x$  és a la primera posició, és a dir,  $x$  és  $v[0]$ . Aleshores només s'entra un cop al bucle, i el cost total és  $\Theta(1)$ .
- (b) Un cas pitjor es dona quan  $x$  no apareix al vector  $v$ . Aleshores es fan  $\Theta(n)$  iteracions del bucle, cadascuna de les quals triga temps  $\Theta(1)$ . El cost total és  $\Theta(n)$ .
- (c) Per inducció.

- *Case base:*  $n = 1$ . Tenim  $\sum_{i=1}^n \frac{i}{2^i} = \frac{1}{2}$ , i  $2 - \frac{n}{2^n} - \frac{1}{2^{n-1}}|_{n=1} = 2 - \frac{1}{2} - 1 = \frac{1}{2}$ .
- *Cas inductiu:*  $n > 1$ . Tenim que  $\sum_{i=1}^{n-1} \frac{i}{2^i} = 2 - \frac{n-1}{2^{n-1}} - \frac{1}{2^{n-2}}$ . Per tant  $\sum_{i=1}^n \frac{i}{2^i} = \frac{n}{2^n} + \sum_{i=1}^{n-1} \frac{i}{2^i} = \frac{n}{2^n} + 2 - \frac{n-1}{2^{n-1}} - \frac{1}{2^{n-2}} = \frac{n}{2^n} + 2 - \frac{2n-2}{2^{n-2}} - \frac{4}{2^n} = 2 + \frac{n-2n+2-4}{2^n} = 2 - \frac{n}{2^n} - \frac{1}{2^{n-1}}$ .

- (d) Si  $x$  és l'element  $v[i]$ , aleshores l'algorisme té cost  $\Theta(i)$ . Per tant el cost mig és

$$\begin{aligned}
 & \sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot \text{Cost}(x = v[i]) = \\
 & \sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot \Theta(i) = \\
 & \Theta\left(\sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot i\right) = \\
 & \Theta\left(\sum_{1 \leq i < n} \text{Prob}(x = v[i]) \cdot i\right) = \\
 & \Theta\left(\text{Prob}(x = v[n-1]) \cdot (n-1) + \sum_{1 \leq i < n-1} \text{Prob}(x = v[i]) \cdot i\right) = \\
 & \Theta\left(\frac{n-1}{2^{n-1}} + \sum_{1 \leq i < n-1} \frac{i}{2^{i+1}}\right) = \\
 & \Theta\left(\frac{n-1}{2^{n-1}} + \frac{1}{2} \cdot \sum_{i=1}^{n-2} \frac{i}{2^i}\right) = \\
 & \Theta\left(\frac{n-1}{2^{n-1}} + \frac{1}{2} \cdot \left(2 - \frac{n-2}{2^{n-2}} - \frac{1}{2^{n-3}}\right)\right) = \\
 & \Theta(1)
 \end{aligned}$$

donat que  $\lim_{n \rightarrow \infty} \frac{1}{2^n} = \lim_{n \rightarrow \infty} \frac{n}{2^n} = 0$ .

### Proposta de solució al problema 3

- (a) Considerem cadascun dels casos:

- Assumim  $a, b$  parells. Tenim  $\gcd(a/2, b/2) \mid a/2$ , i així  $2\gcd(a/2, b/2) \mid a$ . Similment,  $2\gcd(a/2, b/2) \mid b$ . Per tant  $2\gcd(a/2, b/2) \mid \gcd(a, b)$ . Per altra ban-

da, com que  $a$  i  $b$  són parells,  $\gcd(a,b)$  és parell. Però  $\gcd(a,b) \mid a$  implica que  $\gcd(a,b)/2 \mid a/2$ , i similarment  $\gcd(a,b)/2 \mid b/2$ . Per tant  $\gcd(a,b)/2 \mid \gcd(a/2, b/2)$ , i  $\gcd(a,b) \mid 2\gcd(a/2, b/2)$ , d'on finalment  $\gcd(a,b) = 2\gcd(a/2, b/2)$ .

- Assumim  $a$  senar i  $b$  parell. Per una banda  $\gcd(a,b) \mid a$ . Per una altra  $\gcd(a,b) \mid b$ , i com que  $a$  és senar,  $\gcd(a,b) \mid b/2$ . Així doncs tenim que  $\gcd(a,b) \mid \gcd(a, b/2)$ . I com que  $\gcd(a, b/2) \mid a$  i  $\gcd(a, b/2) \mid b$ , tenim  $\gcd(a, b/2) \mid \gcd(a,b)$ . Per tant  $\gcd(a,b) = \gcd(a, b/2)$ .
- Assumim  $a, b$  senars i  $a > b$ . Si  $a$  i  $b$  són senars,  $a - b$  és parell. Per tant  $\gcd(a,b) = \gcd((a-b)/2, b)$  per la pista i l'apartat anterior.

(b) Una possible solució:

```
int gcd(int a, int b) {
 if (a == b) return a;
 if (a == 1 or b == 1) return 1;
 if (a % 2 == 0 and b % 2 == 0) return 2*gcd(a/2, b/2);
 if (a % 2 == 1 and b % 2 == 0) return gcd(a, b/2);
 if (a % 2 == 0 and b % 2 == 1) return gcd(a/2, b);
 else
 if (a > b) return gcd((a-b)/2, b);
 else return gcd(a, (b-a)/2);
}
```

- (c) Un cas pitjor es dona, per exemple, quan  $a$  és una potència de 2 i  $b$  és senar. Aleshores a cada crida recursiva només decreix el primer argument, i  $b$  sempre és el segon argument. Quan el primer argument té  $i$  bits, el cost és  $\Theta(i)$ , per la divisió entre 2. Per tant el cost és  $\sum_{i=1}^n \Theta(i) = \Theta(n^2)$ .



## Solució de l'Examen Parcial EDA

31/03/2016

## Proposta de solució al problema 1

(a)  $\Theta(n^{\log 7})$ 

(b) El teorema mestre de recurrències substractores afirma que si tenim una recurrència de la forma  $T(n) = aT(n - c) + \Theta(n^k)$  amb  $a, c > 0$  i  $k \geq 0$ , llavors

$$T(n) = \begin{cases} \Theta(n^k) & \text{si } a < 1, \\ \Theta(n^{k+1}) & \text{si } a = 1, \\ \Theta(a^{\frac{n}{c}}) & \text{si } a > 1. \end{cases}$$

(c)  $\Theta(n)$ 

## Proposta de solució al problema 2

(a) El cost del programa queda determinat per la suma de costos de dos grups d'instruccions:

- A.** Les instruccions que s'executen a cada iteració del bucle: l'avaluació de la condició  $i \leq n$ , la crida  $f(i)$ , l'avaluació de la condició  $i == p$  i l'increment  $i++$ .
- B.** Les instruccions que s'executen quan la condició de l'if és certa: la crida  $g(n)$  i l'increment  $p *= 2$ .

Comptem per separat la contribució al cost total dels dos grups d'instruccions:

- A.** Si el cost de  $f(m)$  és  $\Theta(m)$ , llavors a la iteració  $i$ -èsima el cost de **A** és  $\Theta(i)$ . Per tant, en total la contribució al cost és  $\sum_{i=1}^n \Theta(i) = \Theta(\sum_{i=1}^n i) = \Theta(n^2)$ .
- B.** Si el cost de  $g(m)$  és  $\Theta(m)$ , llavors cada vegada que la condició de l'if és certa el cost de **B** és  $\Theta(n)$ . Com que això passa  $\Theta(\log n)$  vegades, la contribució al cost total és  $\Theta(n \log n)$ .

En total doncs el cost de  $h(n)$  és  $\Theta(n^2) + \Theta(n \log n) = \Theta(n^2)$ .

(b) Considerem els dos mateixos grups d'instruccions de l'apartat anterior, i de nou comptem per separat la seva contribució al cost total:

- A.** Si el cost de  $f(m)$  és  $\Theta(1)$ , llavors el cost de **A** a cada iteració és  $\Theta(1)$ . Com que es fan  $\Theta(n)$  voltes, el cost en total és  $\Theta(n)$ .
- B.** Si el cost de  $g(m)$  és  $\Theta(m^2)$ , llavors cada vegada que la condició de l'if és certa el cost de **B** és  $\Theta(n^2)$ . Com que això passa  $\Theta(\log n)$  vegades, el cost és  $\Theta(n^2 \log n)$ .

En total doncs el cost de  $h(n)$  és  $\Theta(n) + \Theta(n^2 \log n) = \Theta(n^2 \log n)$ .

**Proposta de solució al problema 3**

(a) La fila  $i$ -èsima ocupa  $i + 1$  caselles ( $0 \leq i < n$ ). En total, l'espai usat és

$$\sum_{i=0}^{n-1} (i+1) = \sum_{j=1}^n j = \Theta(n^2).$$

(b) Una possible solució:

```

int p(int x) { return x*(x+1)/2;}

int misteri(int k, int l, int r) {
 if (l+1 == r) return l;
 int m = (l+r)/2;
 if (p(m) ≤ k) return misteri(k, m, r);
 else return misteri(k, l, m);
}

pair<int,int> fila_columna(int n, int k) {
 if (k < 0 or k ≥ p(n)) return {-1,-1};
 int i = misteri(k, 0, n);
 return {i, k - p(i)};
}

```

(c) Sigui  $C(N)$  el cost en el cas pitjor d'una crida a la funció  $\text{misteri}(k, l, r)$ , on  $N = r - l + 1$ . La recurrència que descriu  $C(N)$  és

$$C(N) = C(N/2) + \Theta(1),$$

ja que es fa una crida recursiva sobre un interval de mida la meitat, més operacions que tenen cost constant. Usant el teorema mestre de recurrències divisores, es té que la solució de la recurrència és  $C(N) = \Theta(\log N)$ .

Per tant, el cost en el cas pitjor d'una crida a la funció  $\text{fila\_columna}(n, k)$  és  $\Theta(\log n)$ .

(d) L'índex  $i$  de la fila buscada és el natural  $0 \leq i < n$  tal que  $p(i) \leq k < p(i+1)$ . El podem calcular resolent l'equació de segon grau  $p(x) = k$  i prenent  $i = \lfloor x \rfloor$ :

```

int p(int x) { return x*(x+1)/2;}

pair<int,int> fila_columna(int n, int k) {
 if (k < 0 or k ≥ p(n)) return {-1,-1};
 int i(floor ((sqrt (1.+8*k) - 1)/2));
 return {i, k - p(i)};
}

```

**Proposta de solució al problema 4**

(a) Demostrem-ho per reducció a l'absurd. Suposem que  $i$  és tal que  $0 \leq i < n-1$  i  $f_A(i+1) < f_A(i)$ . Prenem  $k = i+1$ ,  $j = f_A(i+1)$  i  $l = f_A(i)$ , de forma que  $0 \leq i < k < n$  i  $0 \leq j < l < n$ . Com que  $j = f_A(k)$ , tenim que  $A_{k,j} \leq A_{k,l}$ . I com que  $l = f_A(i)$ , tenim que  $A_{i,l} \leq A_{i,j}$ ; de fet, com que  $j < l$ , ha de ser  $A_{i,l} < A_{i,j}$ . Per tant, sumant tenim que  $A_{i,l} + A_{k,j} < A_{i,j} + A_{k,l}$ . Això contradiu que  $A$  sigui una matriu de Monge.

- (b) Per cada  $i$  parell, es pot calcular la columna on apareix el mínim més a l'esquerra de la fila  $i$  de  $A$  de la manera següent. A partir de la crida recursiva sobre  $B_1$ , tenim la columna  $j_1$  on apareix el mínim més a l'esquerra de la fila  $i$  de  $A$  entre les columnes  $0$  i  $\frac{n}{2} - 1$ . Similarment, a partir de la crida recursiva sobre  $B_2$ , tenim la columna  $j_2$  on apareix el mínim més a l'esquerra de la fila  $i$  de  $A$  entre les columnes  $\frac{n}{2}$  i  $n - 1$ . Per tant,  $f_A(i) = j_1$  si  $A_{i,j_1} \leq A_{i,j_2}$ , i  $f_A(i) = j_2$  altrament.

Per cada  $i$  senar, es determina  $f_A(i)$  recorrent les columnes entre  $f_A(i - 1)$  i  $f_A(i + 1)$  (definim  $f_A(n) = n - 1$  per conveniència notacional), i escollint l'índex on apareix el mínim més a l'esquerra. Això té cost  $\Theta(f_A(i + 1) - f_A(i - 1) + 1)$ . En total:

$$\sum_{i=1, i \text{ senar}}^{n-1} \Theta(f_A(i + 1) - f_A(i - 1) + 1) = \Theta((n - 1) - f_A(0) + \frac{n}{2}) = \Theta(n)$$

ja que  $0 \leq f_A(0) < n$ .

- (c) Sigui  $C(n)$  el cost de l'algorisme proposat per calcular la funció  $f_A$  per a totes les files d'una matriu de Monge  $A$  de mida  $n \times n$ .

A banda de les dues crides recursives del pas (2) sobre matrius de mida  $\frac{n}{2} \times \frac{n}{2}$ , els passos (1) i (3) requereixen temps  $\Theta(n)$  en total. Per tant, la recurrència que descriu el cost és

$$C(n) = 2C(n/2) + \Theta(n).$$

Usant el teorema mestre de recurrències divisores, es té que la solució de la recurrència és  $C(n) = \Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

07/11/2016

## Proposta de solució al problema 1

|                                          | <i>Cas millor</i>  | <i>Cas mig</i>     | <i>Cas pitjor</i>  |
|------------------------------------------|--------------------|--------------------|--------------------|
| (a) Quicksort<br>(amb partició de Hoare) | $\Theta(n \log n)$ | $\Theta(n \log n)$ | $\Theta(n^2)$      |
| Mergesort                                | $\Theta(n \log n)$ | $\Theta(n \log n)$ | $\Theta(n \log n)$ |
| Inserció                                 | $\Theta(n)$        | <b>No ompliu</b>   | $\Theta(n^2)$      |

(b)  $\Theta(\sqrt{n})$ (c)  $\Theta(\sqrt{n} \log n)$ (d)  $\Theta(n)$ (e) L'algorisme de Karatsuba calcula el producte de dos nombres naturals de  $n$  bits en temps  $\Theta(n^{\log_2 3})$ .(f) L'algorisme de Strassen calcula el producte de dues matrius de mida  $n \times n$  en temps  $\Theta(n^{\log_2 7})$ .

## Proposta de solució al problema 2

(a) Una possible solució:

**#include <vector>****using namespace std;**

```
bool dic_search (const vector<int>& a, int l, int r, int x) {
 if (l > r) return false;
 int m = (l+r)/2;
 if (a[m] < x) return dic_search (a, m+1, r, x);
 if (a[m] > x) return dic_search (a, l, m-1, x);
 return true;
}
```

```
bool search (const vector<int>& a, int l, int r, int x) {
 if (l+1 == r) return a[l] == x or a[r] == x;
 int m = (l+r)/2;
 if (a[m] ≥ a[l]) {
 if (a[l] ≤ x and x ≤ a[m]) return dic_search (a, l, m, x);
 else return search (a, m, r, x);
 }
 else {
 if (a[m] ≤ x and x ≤ a[r]) return dic_search (a, m, r, x);
 else return search (a, l, m, x);
 }
}
```

```
bool search (const vector<int>& a, int x) {
 return search (a, 0, a.size()-1, x);
}
```

}

- (b) Sigui  $C(n)$  el cost de tractar un vector de mida  $n$  (sigui amb la funció *search* o amb la funció *dic\_search*) en el cas pitjor (per exemple, quan l'element  $x$  no apareix a la seqüència). A banda d'operacions de cost constant (càlculs aritmètics, comparacions i assignacions entre enters, accessos a vectors), es fa exactament una crida sobre un vector de mida la meitat de la de l'entrada. Així doncs, el cost ve determinat per la recurrència:

$$C(n) = C(n/2) + \Theta(1),$$

que, pel teorema mestre de recurrències divisores, té solució  $C(n) = \Theta(\log(n))$ .

### Proposta de solució al problema 3

- (a) Calcula  $m^n$ .
- (b) El cost de la funció ve determinat pel cost del bucle. Cada iteració requereix temps  $\Theta(1)$ , ja que només s'hi duen a terme operacions aritmètiques i assignacions d'enters. Per tant, el cost és proporcional al nombre d'iteracions. Observem que si  $y$  és parell, aleshores  $y$  es redueix a la meitat. I si  $y$  és un senar amb  $y > 1$ , a la següent iteració es considera  $y - 1$ , que és parell, i llavors es redueix a la meitat. Així doncs, si  $n \geq 1$  el nombre de voltes està entre  $1 + \lfloor \log(n) \rfloor$  i  $1 + 2\lfloor \log(n) \rfloor$ . En conclusió, el cost és  $\Theta(\log(n))$ .

### Proposta de solució al problema 4

- (a) Tenim que  $\phi - 1 = \frac{\sqrt{5}+1}{2} - 1 = \frac{\sqrt{5}-1}{2}$ , i aleshores

$$\phi \cdot (\phi - 1) = \frac{\sqrt{5}+1}{2} \cdot \frac{\sqrt{5}-1}{2} = \frac{(\sqrt{5}+1)(\sqrt{5}-1)}{4} = \frac{5-1}{4} = 1$$

De forma que efectivament  $\phi^{-1} = \phi - 1$ .

- (b) Per inducció sobre  $n$ :

- **Cas base  $n = 0$ :** per una banda  $F(0) = 0$ , i per una altra

$$F(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}} \Big|_{n=0} = \frac{1-1}{\sqrt{5}} = 0.$$

- **Cas base  $n = 1$ :** per una banda  $F(1) = 1$ , i per una altra

$$F(n) = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}} \Big|_{n=1} = \frac{\phi + \phi^{-1}}{\sqrt{5}} = \frac{\sqrt{5}}{\sqrt{5}} = 1.$$

- **Cas inductiu  $n > 1$ :** per hipòtesi d'inducció,

$$\begin{aligned} F(n) &= F(n-2) + F(n-1) = \frac{\phi^{n-2} - (-\phi)^{-(n-2)}}{\sqrt{5}} + \frac{\phi^{n-1} - (-\phi)^{-(n-1)}}{\sqrt{5}} = \\ &= \frac{\phi^{n-1}(\phi^{-1} + 1) - (-\phi)^{-(n-1)}(-\phi + 1)}{\sqrt{5}} = \frac{\phi^{n-1} \cdot \phi - (-\phi)^{-(n-1)}(-\phi^{-1})}{\sqrt{5}} = \\ &= \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}} \end{aligned}$$

(c) Tenim que

$$\lim_{n \rightarrow +\infty} \frac{F(n)}{\phi^n} = \lim_{n \rightarrow +\infty} \frac{\frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}}{\phi^n} = \lim_{n \rightarrow +\infty} \frac{1 - \left(\frac{-1}{\phi^2}\right)^n}{\sqrt{5}} = \frac{1}{\sqrt{5}}$$

donat que  $\phi > 1$ ,  $\phi^2 > 1$  i  $\lim_{n \rightarrow +\infty} \left(\frac{-1}{\phi^2}\right)^n = 0$ . Per tant  $F(n) = \Theta(\phi^n)$ .

## Solució de l'Examen Parcial EDA

20/04/2017

## Proposta de solució al problema 1

- (a) L'algorisme d'ordenació per inserció té cost  $\Theta(n)$  quan el vector està ordenat, i cost  $\Theta(n^2)$  quan està ordenat del revés. Així doncs, el cost mig és:

$$\left(1 - \frac{\log n}{n}\right)\Theta(n) + \frac{\log n}{n}\Theta(n^2) = \Theta(n - \log n + n \log n) = \Theta(n \log n)$$

- (b) La funció retorna si  $n$  és un nombre primer. El cost està determinat pel bucle, que fa  $O(\sqrt{n})$  iteracions, cadascuna de les quals de cost constant. Així doncs, el cost és  $O(\sqrt{n})$ .
- (c) Cadascun dels nombres que es multipliquen a  $n!$  són menors o iguals que  $n$ . Com que se'n multipliquen  $n$ , es té  $n! \leq n^n$  per tot  $n \geq 1$ . D'aquí  $n! = O(n^n)$ , prenent  $n_0 = 1$  i  $C = 1$ .
- (d) Llevat de 1, tots els nombres que es multipliquen a  $n!$  són majors o iguals que 2. Com que se'n multipliquen  $n - 1$ , es té  $n! \geq 2^{n-1} = \frac{1}{2} \cdot 2^n$  per tot  $n \geq 1$ . D'aquí  $n! = \Omega(2^n)$ , prenent  $n_0 = 1$  i  $C = \frac{1}{2}$ .

## Proposta de solució al problema 2

- (a) Una possible solució:

```

int top_rec (const vector<int>& a, int l, int r) {
 if (l + 1 ≥ r) {
 if (a[l] < a[r]) return r;
 else return l;
 }
 int m = (l+r)/2;
 if (a[m-1] > a[m]) return top_rec(a, l, m-1);
 if (a[m+1] > a[m]) return top_rec(a, m+1, r);
 return m;
}

int top(const vector<int>& a) {
 return top_rec(a, 0, a.size()-1);
}

```

Sigui  $C(n)$  el cost en temps en el cas pitjor de la funció `top_rec` sobre un vector de mida  $n$ . En el cas pitjor (per exemple, quan el cim és en un extrem del vector) es farà una crida recursiva sobre un subvector de mida  $n/2$ , a més d'operacions de cost constant (càlculs aritmètics, comparacions i assignacions entre enters, accessos a vectors). Així doncs tenim la recurrència  $C(n) = C(n/2) + \Theta(1)$ , que pel teorema mestre de recurrències divisores té solució  $C(n) = \Theta(\log n)$ .

- (b) Una possible solució que usa la funció `int top(const vector<int>& a)` de l'apartat anterior i la funció `binary_search` de la STL:

```

bool search(const vector<int>& a, int x) {
 int t = top(a);

```

```

int n = a.size ();
if (binary_search (a.begin (), a.begin () + t, x)) return true;
if (binary_search (a.rbegin (), a.rbegin () + n - t, x)) return true;
return false;
}

```

En una altra possible solució, la funció *search* anterior es pot reemplaçar per:

```

bool bin_search_inc (const vector<int>& a, int l, int r, int x) {
 if (l > r) return false;
 int m = (l+r)/2;
 if (a[m] < x) return bin_search_inc (a, m+1, r, x);
 if (a[m] > x) return bin_search_inc (a, l, m-1, x);
 return true;
}

bool bin_search_dec (const vector<int>& a, int l, int r, int x) {
 if (l > r) return false;
 int m = (l+r)/2;
 if (a[m] < x) return bin_search_dec (a, l, m-1, x);
 if (a[m] > x) return bin_search_dec (a, m+1, r, x);
 return true;
}

bool search (const vector<int>& a, int x) {
 int t = top(a);
 int n = a.size ();
 if (bin_search_inc (a, 0, t-1, x)) return true;
 if (bin_search_dec (a, t, n-1, x)) return true;
 return false;
}

```

Quan la funció *search* busca en un vector de mida  $n$ , a més de cridar la funció *top*, també es fan una o dues cerques dicotòmiques, cadascuna de les quals té cost  $O(\log n)$ , i operacions de cost constant. Per tant, el cost de *search* és  $O(\log n) + O(\log n) + O(1) = O(\log n)$ . A més, si per exemple el cim de la seqüència és en un extrem del vector, aleshores el cost és  $\Theta(\log n) + O(\log n) + O(1) = \Theta(\log n)$ . Per tant, el cost en el cas pitjor és  $\Theta(\log n)$ .

### Proposta de solució al problema 3

(a) Després de  $m$  crides a la funció *reserve* sobre un vector inicialment buit, la capacitat del vector és de  $C(m) = Am$  elements. Per tant, el nombre de crides fetes a *reserve* després de  $n$  crides a *push\_back* és el menor  $m$  tal que  $Am \geq n$ , és a dir,  $\lceil \frac{n}{A} \rceil$ . Com que  $A$  és constant,  $m$  és  $\Theta(n)$ .

(b) Per inducció.

- **Cas base:**  $m = 0$ . Tenim que  $\frac{BA^m - B}{A - 1} \Big|_{m=0} = \frac{B - B}{A - 1} = 0 = C(0)$ .



- **Cas inductiu:** suposant que és cert per a  $m$ , vegem que és cert per a  $m + 1$ . Per hipòtesi d'inducció es té  $C(m) = \frac{BA^m - B}{A - 1}$ . Aleshores:

$$C(m + 1) = AC(m) + B = A \frac{BA^m - B}{A - 1} + B = \frac{BA^{m+1} - AB + AB - B}{A - 1} = \frac{BA^{m+1} - B}{A - 1}$$

- (c) Després de  $m$  crides a la funció *reserve* sobre un vector inicialment buit, usant l'apartat anterior tenim que la capacitat del vector és de  $C(m) = \frac{BA^m - B}{A - 1}$  elements. Per tant, el nombre de crides fetes a *reserve* després de  $n$  crides a *push\_back* és el menor  $m$  tal que  $\frac{BA^m - B}{A - 1} \geq n$ , és a dir,  $\lceil \log_A(\frac{n(A-1)+B}{B}) \rceil$ . Com que  $A$  i  $B$  són constants,  $m$  és  $\Theta(\log n)$ .

#### Proposta de solució al problema 4

- (a) Una possible solució:

```
int stable_partition (int x, vector<int>& a) {
 int n = a.size ();
 vector<int> w(n);
 int i = 0;
 for (int y : a)
 if (y ≤ x) {
 w[i] = y;
 ++i;
 }
 int r = i - 1;
 for (int y : a)
 if (y > x) {
 w[i] = y;
 ++i;
 }
 for (int k = 0; k < n; ++k)
 a[k] = w[k];

 return r;
}
```

El cost en temps és  $\Theta(n)$ , ja que es fan 3 recorreguts sobre el vector, cada iteració dels quals només requereix temps constant. El cost en espai de la memòria auxiliar està dominat pel vector  $w$ , que té mida  $n$ . Així, el cost en espai és  $\Theta(n)$ .

- (b) Transposa els dos subvectors de  $a$  de  $l$  a  $p$  i de  $p + 1$  a  $r$ : si abans de la crida  $a[l..r]$  és  $A_l, \dots, A_p, A_{p+1}, \dots, A_r$ , després de la crida  $a[l..r]$  és  $A_{p+1}, \dots, A_r, A_l, \dots, A_p$ .

- (c) Solució:

```
int stable_partition (int x, vector<int>& a) {
 return stable_partition_rec (x, a, 0, a.size () - 1);
}

int stable_partition_rec (int x, vector<int>& a, int l, int r) {
 if (l == r) {
 if (a[l] ≤ x) return l;
 else return l - 1;
 }
}
```

```
}
int m = (l+r)/2;
int p = stable_partition_rec (x, a, l, m);
int q = stable_partition_rec (x, a, m+1, r);
mystery(a, p+1, m, q);
return p+q-m;
}
```

Sigui  $C(n)$  el cost de la funció *stable\_partition\_rec* sobre un vector de mida  $n = r - l + 1$  en el cas pitjor. Per una banda es fan dues crides recursives sobre vectors de mida  $n/2$ . Per una altra, el cost del treball no recursiu està dominat per la funció *mystery*, que triga temps lineal en la mida del vector que transposa. Usant la hipòtesi de l'enunciat (que es dóna, per exemple, en un vector en què es van alternant successivament elements més petits i més grans que  $x$ ), aquest vector té mida  $\Theta(n)$ . Així doncs tenim la recurrència  $C(n) = 2C(n/2) + \Theta(n)$ , que pel teorema mestre de recurrències divisores té solució  $\Theta(n \log n)$ . En conclusió, el cost de la funció *stable\_partition* sobre un vector de mida  $n$  en el cas pitjor és  $\Theta(n \log n)$ .

## Solució de l'Examen Parcial EDA

06/11/2017

## Proposta de solució al problema 1

- (a) El cost és  $\Theta(n\sqrt{n})$ .
- (b) El cost tant en el cas pitjor com en el cas mig és  $\Theta(n)$ .
- (c)  $\Theta(\sqrt{n})$ .
- (d)  $\Theta(n^2 \log n)$ .
- (e)  $\Theta(2^n)$ .

## Proposta de solució al problema 2

- (a) El valor  $b[y]$  compta el nombre d'elements de la seqüència  $a$  que són més petits o iguals que  $y$ .
- (b) Una possible solució:

```

int median(int n, const vector<int>& b) {
 int l = 0;
 int r = b.size() - 1;
 while (l < r) {
 int q = (l+r)/2;
 if (b[q] < b[(l+r)/2]) l = q;
 else r = q;
 }
 return r;
}

```

- (c) A cada volta del bucle, el valor  $r - l$  es redueix per la meitat. Com que inicialment  $l = 0$  i  $r = m$ , es fan  $\Theta(\log m)$  voltes. I com que cada volta té cost constant (operacions aritmètiques d'enters, accessos a vectors, etc.), concloem que el cost de la funció és  $\Theta(\log m)$ .

## Proposta de solució al problema 3

- (a) Una possible solució:

```

complex operator*(const complex& a, const complex& b) {
 return {a.real * b.real - a.imag * b.imag, a.real * b.imag + a.imag * b.real};
}

complex exp(complex z, int n) {
 if (n == 0) return {1, 0};
 complex x = exp(z, n/2);
 complex y = x*x;
 if (n % 2 == 1) y = y*z;
 return y;
}

```

- (b) El cost  $C(n)$  de la funció recursiva *exp* en funció de  $n$  ve determinat per la recurrència  $C(n) = C(n/2) + \Theta(1)$ : es fa una sola crida recursiva sobre un problema de mida  $n/2$ , i la resta d'operacions tenen cost constant. Aplicant el teorema mestre de recurrències divisores, tenim que la solució és  $C(n) = \Theta(\log n)$  tal com es demanava.

**Proposta de solució al problema 4**

- (a) Tenim que

$$U(m) = T(2^m) = T(2^m/2) + \log(2^m) = T(2^{m-1}) + \log(2^m) = U(m-1) + m$$

- (b) Aplicant el teorema mestre de recurrències substractores per a resoldre la recurrència  $U(m) = U(m-1) + m$ , tenim que  $U(m) = \Theta(m^2)$ .
- (c) Com que  $U(m) = \Theta(m^2)$  i  $U(m) = T(2^m)$ , tenim que  $T(n) = T(2^{\log n}) = U(\log n) = \Theta((\log n)^2)$ .

**Solució de l'Examen Parcial EDA****19/04/2018****Proposta de solució al problema 1**(a) La funció  $g$  creix més de pressa:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{n^{n^2}}{2^{2^n}} = \lim_{n \rightarrow \infty} 2^{\log(\frac{n^{n^2}}{2^{2^n}})} = 2^{\lim_{n \rightarrow \infty} \log(\frac{n^{n^2}}{2^{2^n}})} = 0$$

perquè

$$\lim_{n \rightarrow \infty} \log\left(\frac{n^{n^2}}{2^{2^n}}\right) = \lim_{n \rightarrow \infty} (\log(n^{n^2}) - \log(2^{2^n})) = \lim_{n \rightarrow \infty} (n^2 \log n - 2^n) = -\infty$$

(b) El cost de l'algorisme en el cas mig és  $\Theta(n^2)$ :

$$\frac{1}{n^3} \cdot \Theta(n^4) + \frac{1}{n} \cdot \Theta(n^3) + \left(1 - \frac{1}{n^3} - \frac{1}{n}\right) \cdot \Theta(n) = \Theta(n) + \Theta(n^2) + \Theta(n) = \Theta(n^2)$$

(c) Usant el teorema mestre de recurrències divisores tenim que  $a = 2$ ,  $b = 4$ ,  $\alpha = \log_4 2 = \frac{1}{2}$  i  $k = \frac{1}{2}$ , de forma que  $T(n) = \Theta(\sqrt{n} \cdot \log n)$ .**Proposta de solució al problema 2**(a) El cas base per a  $k = 0$  és cert:  $A^0 \cdot x_0 = x_0 = x(0)$ . Pel cas inductiu, quan  $k > 0$ , tenim per hipòtesi d'inducció que  $x(k-1) = A^{k-1} \cdot x_0$ . Per tant,  $x(k) = A \cdot x(k-1) = A \cdot (A^{k-1} \cdot x_0) = (A \cdot A^{k-1}) \cdot x_0 = A^k \cdot x_0$ .(b) Es calcula en primer lloc  $A^k$  usant l'algorisme d'exponenciació ràpida en temps  $\Theta(\log k)$  (ja que la  $n$  és constant). Després es retorna el resultat de multiplicar  $A^k$  per  $x_0$ , que es pot fer en temps  $\Theta(1)$  (de nou, perquè la  $n$  és constant). El cost en temps de l'algorisme és doncs  $\Theta(\log k)$ .**Proposta de solució al problema 3**(a)  $x = x_2 \cdot 3^{2n/3} + x_1 \cdot 3^{n/3} + x_0$ .(b)  $x \cdot y = x_2 y_2 \cdot 3^{4n/3} + (x_1 y_2 + x_2 y_1) \cdot 3^{3n/3} + (x_0 y_2 + x_1 y_1 + x_2 y_0) \cdot 3^{2n/3} + (x_1 y_0 + x_0 y_1) \cdot 3^{n/3} + x_0 y_0$ .(c)  $x \cdot y =$ 

$$x_2 y_2 \cdot 3^{4n/3}$$

$$+ (x_1 y_2 + x_2 y_1) \cdot 3^{3n/3}$$

$$+ ((x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2) - x_2 y_2 - (x_1 y_2 + x_2 y_1) - (x_1 y_0 + x_0 y_1) - x_0 y_0) \cdot 3^{2n/3}$$

$$+ (x_1 y_0 + x_0 y_1) \cdot 3^{n/3}$$

$$+ x_0 y_0$$

Es fan servir 7 productes.

- (d) Per a calcular el producte de  $x$  i  $y$  de  $n$  dígit, l'algorisme calcula recursivament  $(x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2)$ ,  $x_2y_2$ ,  $x_1y_2 + x_2y_1$ ,  $x_1y_0 + x_0y_1$  i  $x_0y_0$ , que són productes de nombres de  $n/3$  dígit. Llavors es calcula  $x \cdot y$  usant l'equació de l'apartat anterior. Com que els nombres es representen en base 3, multiplicar per una potència de 3 és afegir zeros a la dreta i es pot fer en temps  $\Theta(n)$ . Les sumes involucrades també es poden fer en temps  $\Theta(n)$ . Així doncs el cost  $T(n)$  satisfà la recurrència  $T(n) = 7T(n/3) + \Theta(n)$ , que té solució  $\Theta(n^{\log_3 7})$ .

#### Proposta de solució al problema 4

- (a) Definim la funció  $U(m) = T(b^m)$ . Llavors  $T(n) = U(\log_b(n))$ . A més, tenim:

$$\begin{aligned} U(m) = T(b^m) &= T(b^m/b) + \Theta(\log^k(b^m)) = T(b^{m-1}) + \Theta(m^k \log^k(b)) = \\ &= T(b^{m-1}) + \Theta(m^k) = U(m-1) + \Theta(m^k) \end{aligned}$$

El teorema mestre de recurrències substractores afirma que si tenim una recurrència de la forma  $U(m) = U(m-c) + \Theta(m^k)$  amb  $c > 0$  i  $k \geq 0$ , llavors  $U(m) = \Theta(m^{k+1})$ . Per tant la solució a la recurrència de l'enunciat és  $T(n) = \Theta((\log_b(n))^{k+1}) = \Theta(\log^{k+1} n)$ .

- (b) Una possible solució:

```
bool search(const vector<int>& a, int x, int l, int r) {
 if (l == r) return x == a[l];
 int m = (l+r)/2;
 auto beg = a.begin();
 if (a[m] < a[m+1])
 return search(a, x, m+1, r) or binary_search(beg + l, beg + m + 1, x);
 else
 return search(a, x, l, m) or binary_search(beg + m+1, beg + r + 1, x);
}

bool search(const vector<int>& a, int x) {
 return search(a, x, 0, a.size()-1);
}
```

- (c) El cas pitjor es dona per exemple quan  $x$  no apareix a  $a$ . En aquesta situació el cost  $T(n)$  ve descrit per la recurrència  $T(n) = T(n/2) + \Theta(\log n)$ , perquè es fa una crida recursiva sobre un vector de mida  $\frac{n}{2}$  i el cost del treball no recursiu està dominat per la cerca binària, que té cost  $\Theta(\log(\frac{n}{2})) = \Theta(\log(n))$ . Aplicant el primer apartat tenim que la solució és  $T(n) = \Theta(\log^2(n))$ .

## Solució de l'Examen Parcial EDA

05/11/2018

## Proposta de solució al problema 1

(a) Les respostes:

|      | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) | (10) |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| CERT | X   |     | X   | X   |     | X   |     |     | X   | X    |
| FALS |     | X   |     |     | X   |     | X   | X   |     |      |

## Proposta de solució al problema 2

- (a) El cost en el cas pitjor és  $\Theta(n^3)$ . El cas pitjor es dona sempre.
- (b) El cost en el cas millor és  $\Theta(n^2)$ . El cas millor es dona, per exemple, quan les dues matrius només tenen *true* en els seus coeficients.

El cost en el cas pitjor és  $\Theta(n^3)$ . El cas pitjor es dona, per exemple, quan les dues matrius només tenen *false* en els seus coeficients.

- (c) La funció considera les matrius booleanes com a matrius d'enters, en què *false* s'interpreta com 0 i *true* com 1. Aleshores s'aplica l'algorisme de producte de matrius d'Strassen. Sigui  $M$  la matriu resultat. El coeficient de la fila  $i$ -èsima i columna  $j$ -èsima de  $M$  és

$$m_{ij} = \sum_{k=0}^{n-1} (a_{ik} \cdot b_{kj}).$$

Com que els coeficients de les matrius d'entrada són 0 o 1, el producte com a nombres enters és el mateix que l'operació  $\wedge$  lògica. De forma que  $m_{ij}$  compta el nombre de parells  $(a_{ik}, b_{kj})$  on ambdós  $a_{ik}$  i  $b_{kj}$  són certs a la vegada. Així doncs, per obtenir el producte lògic només cal definir  $p_{ij}$  com 1 si  $m_{ij} > 0$ , i 0 altrament. El cost està dominat per l'algorisme d'Strassen, que té cost  $\Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$ .

## Proposta de solució al problema 3

- (a) Donat un enter  $n \geq 0$ , la funció *mystery* calcula  $\lfloor \sqrt{n} \rfloor$ .
- (b) En primer lloc trobem el cost  $C(N)$  de la funció *mystery\_rec* en termes de la mida  $N = r - l + 1$  de l'interval  $[l, r]$ . Aquest cost segueix la recurrència  $C(N) = C(N/2) + \Theta(1)$ , ja que a cada crida es fa una crida recursiva sobre un interval amb la meitat d'elements, i treball addicional no recursiu de cost constant. D'acord amb el teorema mestre de recurrències divisores, la solució d'aquesta recurrència és  $\Theta(\log N)$ . Com que *mystery*( $n$ ) consisteix en cridar *mystery\_rec*( $n, 0, n+1$ ), podem concloure que el cost de *mystery*( $n$ ) és  $\Theta(\log n)$ .

## Proposta de solució al problema 4

- (a)  $\Theta(n)$
- (b)  $\Theta(n^2)$
- (c)  $\Theta(n \log n)$
- (d)  $\Theta(n \log n)$

(e) Una possible solució:

```
void my_sort(vector<int>& v) {
 int n = v.size ();
 double lim = n * log(n);
 int c = 0;
 for (int i = 1; i < n; ++i) {
 int x = v[i];
 int j;
 for (j = i; j > 0 and v[j - 1] > x; --j) {
 v[j] = v[j - 1];
 ++c;
 }
 v[j] = x;
 if (c > lim) {
 merge_sort(v);
 return;
 }
 }
}
```

(f) En primer lloc observem que, com que el bucle **for** extern fa  $n - 1$  voltes, cadascuna de les quals té cost  $\Omega(1)$ , el cost de tota execució és  $\Omega(n)$ .

Si per exemple el vector està ja ordenat de forma creixent, aleshores *my\_sort* es comporta com l'ordenació per inserció: no s'entra mai al bucle **for** intern,  $c$  és sempre 0, i no es crida *merge\_sort*. Com que cada volta del bucle **for** extern té cost constant, el cost en aquest cas és  $\Theta(n)$ . De forma que el cost de *my\_sort* en el cas millor és  $\Theta(n)$ .

Per veure el cost en el cas pitjor, distingim dos casos:

- Suposem que no s'arriba a cridar *merge\_sort*. Aleshores el cost és proporcional al valor final de la variable  $c$ . Com que no es crida *merge\_sort*, tenim que  $c \leq n \ln n$ , i el cost és  $O(n \ln n) = O(n \log n)$ .
- Suposem que es crida *merge\_sort*. Si es crida al final de la primera volta del bucle **for** extern, aleshores el cost és  $O(n)$  del bucle **for** intern més  $\Theta(n \log n)$  del *merge\_sort*. En total, el cost és  $\Theta(n \log n)$ .

Si *merge\_sort* es crida al final de la segona, o tercera, etc. volta del bucle **for** extern, aleshores a la iteració anterior del bucle **for** extern no s'ha cridat *merge\_sort*. A més, en la darrera iteració  $c$  com a molt pot haver augmentat en  $i$ , de forma que en el moment de cridar *merge\_sort*, tenim que  $n \ln n < c \leq i + n \ln n \leq n + n \ln n \leq n \ln n + n \ln n = 2n \ln n$  (per  $n$  prou gran), i per tant  $c = \Theta(n \log n)$ . Com que el cost de *merge\_sort* és  $\Theta(n \log n)$ , en total el cost és  $\Theta(n \log n)$ .

El cas pitjor es dona per exemple quan el vector està ordenat al revés, és a dir, de forma decreixent. Com que en aquest cas l'ordenació per inserció té cost  $\Theta(n^2)$ , en algun moment de l'execució de *my\_sort* es cridarà *merge\_sort*, i pel raonament anterior el cost serà  $\Theta(n \log n)$ .



## Solució de l'Examen Parcial EDA

25/04/2019

## Proposta de solució al problema 1

(a) Les respostes:

(1)  $f = O(g) : \text{si } \alpha > 1.$

(2)  $f = \Omega(g) : \text{si } \alpha \leq 1.$

(3)  $g = O(f) : \text{si } \alpha \leq 1.$

(4)  $g = \Omega(f) : \text{si } \alpha > 1.$

(b)  $T(n) = \Theta(2^{\frac{n}{2}}) = \Theta(\sqrt{2}^n)$

(c)  $T(n) = \Theta(n \log n)$

(d)  $\Theta(n \log n)$

(e)  $\Theta(n \log n)$

## Proposta de solució al problema 2

(a) Una crida  $mystery(v, 0, v.size() - 1, m)$  permuta els elements de  $v$  de forma que (almenys) les  $m$  primeres posicions de  $v$  estiguin ocupades pels  $m$  elements més petits, ordenats de forma creixent.

(b) Quan es crida  $mystery(v, 0, n-1, n)$ , primer s'executa  $\mathbf{int} \ q = \mathit{partition}(v, l, r)$  amb  $l = 0$  i  $r = n-1$ , i es triga  $\Theta(n)$ . A més, com que el vector és d'enters diferents ordenats creixentment i la funció  $\mathit{partition}$  pren com a pivot l'element de més a l'esquerra, tindrem  $q = 0$ . De forma que la crida  $mystery(v, l, q, m)$  es fa amb  $q = l = 0$  i per tant triga temps constant. A més, llavors  $p = 1$ . Si  $n > 1$  finalment es fa una crida recursiva  $mystery(v, q+1, r, m-p)$ , on  $q+1 = 1, r = n-1$  i  $m-p = n-1$ .

A la seva vegada, en executar aquesta crida el cost de  $\mathbf{int} \ q = \mathit{partition}(v, l, r)$  és  $\Theta(n-1)$ , de nou el cost de  $mystery(v, l, q, m)$  és  $\Theta(1)$ , i si  $n-1 > 1$  es fa de nou una crida recursiva  $mystery(v, q+1, r, m-p)$ , on  $q+1 = 2, r = n-1$  i  $m-p = n-2$ .

Repetint l'argument, veiem que el cost acumulat és

$$\Theta(n) + \Theta(n-1) + \dots + \Theta(1) = \Theta\left(\sum_{i=1}^n i\right) = \Theta(n^2).$$

## Proposta de solució al problema 3

(a) Una possible solució:

```
vector<bool> prod(const vector<bool>& x, const vector<bool>& y) {
 if (x.size() == 0 or y.size() == 0) return vector<bool>();
 vector<bool> z = twice(twice(prod(half(x), half(y))));
```

```

vector<bool> one = vector<bool>(1, 1);

if (x.back() == 0 and y.back() == 0) return z;
else if (x.back() == 1 and y.back() == 0) return sum(z, y);
else if (y.back() == 1 and x.back() == 0) return sum(z, x);
else {
 vector<bool> x2 = twice(half(x));
 vector<bool> y2 = twice(half(y));
 return sum(sum(sum(z, x2), y2), one);
}

```

Sigui  $T(n)$  el cost de  $prod(x, y)$  si  $n = x.size() = y.size()$ . Es fa una única crida recursiva sobre vectors de mida  $n - 1$ . El treball no recursiu té cost  $\Theta(n)$ . Per tant el cost segueix la recurrència

$$T(n) = T(n - 1) + \Theta(n).$$

D'acord amb el teorema mestre de recurrències subtractives, la solució d'aquesta recurrència es comporta asimptòticament com  $\Theta(n^2)$ . Per tant, el cost és  $\Theta(n^2)$ .

(b) L'algorisme de Karatsuba, que té cost  $\Theta(n^{\log 3})$ .

#### Proposta de solució al problema 4

(a) Una possible forma de completar el codi:

```

bool search1(int x, const vector<vector<int>>& A, int i, int j, int n) {
 if (n == 1) return A[i][j] == x;
 int mi = i + n/2 - 1;
 int mj = j + n/2 - 1;

 if (A[mi][mj] < x) return
 search1(x, A, mi+1, j, n/2) or
 search1(x, A, mi+1, mj+1, n/2) or
 search1(x, A, i, mj+1, n/2);

 if (A[mi][mj] > x) return
 search1(x, A, mi+1, j, n/2) or
 search1(x, A, i, j, n/2) or
 search1(x, A, i, mj+1, n/2);

 return true;
}

```

Observem que el resultat de cridar  $search1(x, A, 0, 0, N)$  és **true** si i només si  $x$  apareix a  $A$ .

Per analitzar el cost d'aquesta crida, en primer lloc estudiem el cas general de cridar  $search1(x, A, i, j, n)$  en funció de  $n$ . Sigui  $T(n)$  el cost en el cas pitjor d'aquesta crida. Com que en el cas pitjor es fan 3 crides amb darrer paràmetre  $n/2$  i el cost del treball no recursiu és constant, tenim que se segueix la recurrència:

$$T(n) = 3T(n/2) + \Theta(1)$$

Aplicant el teorema mestre de recurrències divisores, tenim  $T(n) = \Theta(n^{\log_2 3})$ .

Així doncs, el cost de cridar `search1(x, A, 0, 0, N)` és  $\Theta(N^{\log_2 3})$ .

(b) És correcta. Vegem que el bucle manté el següent invariant: si  $x$  apareix a  $A$ , llavors ho fa entre la fila 0 i la  $i$  (incloses), i entre la columna  $j$  i la  $N - 1$  (incloses). En començar el bucle, l'invariant és cert. I a cada volta es manté:

- Si  $A[i][j] > x$  aleshores  $x$  no pot aparèixer a la fila  $i$ : de l'invariant tenim que si  $x$  apareix ha de ser en una columna de la  $j$  a la  $N - 1$ . Però si  $j \leq k < N$  llavors  $A[i][k] \geq A[i][j] > x$ .
- Si  $A[i][j] < x$  aleshores  $x$  no pot aparèixer a la columna  $j$ : de l'invariant tenim que si  $x$  apareix ha de ser en una fila de la 0 a la  $i$ . Però si  $0 \leq k \leq i$  llavors  $A[k][j] \leq A[i][j] < x$ .

Per últim, si el programa retorna **true** pel **return** de dins del bucle, la resposta és correcta perquè  $A[i][j] = x$ . Si retorna **false** pel **return** de fora, llavors  $i < 0$  o  $j \geq N$ . En qualsevol cas, de l'invariant deduïm que  $x$  no apareix a  $A$ .

(c) A cada volta o bé  $i$  es decrementa en 1, o bé  $j$  s'incrementa en 1, o bé es retorna. A més, inicialment  $i$  val  $N - 1$ , i com a mínim val 0 abans de sortir del bucle. Similarment, al principi  $j$  val 0 i com a molt val  $N - 1$  abans de sortir del bucle. Com que en el cas pitjor es poden fer  $2N - 1$  voltes i cadascuna té cost  $\Theta(1)$ , el cost és  $\Theta(N)$ .

## Solució de l'Examen Parcial EDA

11/11/2019

## Proposta de solució al problema 1

(a)  $\Theta(\sqrt{n} \log n)$ 

(b) Calculem:

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log(\log n^2)}{\log n} &= \lim_{n \rightarrow \infty} \frac{\log(2 \log n)}{\log n} = \lim_{n \rightarrow \infty} \frac{\log 2 + \log(\log n)}{\log n} = \\ &= \lim_{n \rightarrow \infty} \frac{\log 2}{\log n} + \lim_{n \rightarrow \infty} \frac{\log(\log n)}{\log n} = \lim_{n \rightarrow \infty} \frac{\log(\log n)}{\log n} \end{aligned}$$

Fem el canvi de variable  $n = 2^m$  i obtenim que l'anterior és igual a

$$\lim_{m \rightarrow \infty} \frac{\log(\log 2^m)}{\log 2^m} = \lim_{m \rightarrow \infty} \frac{\log m}{m} = 0$$

Per tant únicament és cert que  $\log(n) \in \Omega(\log(\log(n^2)))$ 

## Proposta de solució al problema 2

(a) Retorna  $f \circ g$ , la composició de  $f$  amb  $g$ . El cost de *misteri* és el de la funció auxiliar *misteri\_aux*, que ve descrit per la recurrència  $T(n) = T(n-1) + \Theta(1)$ , que té com a solució asimptòtica  $T(n) \in \Theta(n)$ .

(b) Retorna  $f^k$ . És a dir, una funció tal que  $f^k(x) = \underbrace{f(f(\dots(f(x))))}_k$ . En funció de  $k$ , el seu cost ve donat per la recurrència  $T(k) = T(k-1) + \Theta(1)$ , que té com a solució  $\Theta(k)$ .

(c)

```
vector<int> misteri_2_quick(const vector<int>&f, int k) {
 if (k == 0) {
 vector<int> r(f.size());
 for (uint i = 0; i < f.size(); ++i) r[i] = i;
 return r;
 }
 else if (k%2 == 0) {
 vector<int> aux = misteri_2_quick(f, k/2);
 return misteri(aux, aux);
 }
 else {
 vector<int> aux = misteri_2_quick(f, k/2);
 return misteri(f, misteri(aux, aux));
 }
}
```

La recurrència que descriu el cost en temps d'aquesta funció és  $T(k) = T(k/2) + \Theta(1)$ , que té com a solució asimptòtica  $\Theta(\log k)$ .

**Proposta de solució al problema 3**

- (a) És fàcil veure que la funció *max\_suma* essencialment implementa una ordenació per selecció, que sabem que té cost en cas pitjor de  $\Theta(m^2)$ . L'única diferència és la línia on actualitzem *suma*, que triga temps constant i només s'executa *m* vegades. Per tant el cost total és  $\Theta(m^2) + \Theta(m) = \Theta(m^2)$ .
- (b) Si entenem el codi anterior ens podem adonar que ordena el vector de major a menor i agrupa els enters consecutivament de dos en dos seguint aquest ordre. Per millorar l'eficiència, només cal ordenar el vector amb un *merge sort*, de manera que el cost sigui  $\Theta(m \log m)$ , i agrupar els enters consecutivament de dos en dos. El cost asimptòtic en temps seria de  $\Theta(m \log m)$ .
- (c) Assumim que  $x_0$  i  $x_1$  són els dos nombres més grans de *S* i considerem una expressió que conté els productes  $x_0 * y$  i  $x_1 * z$ , per certs  $y, z \in S$ . El que farem és reemplaçar aquests dos productes per  $x_0 * x_1$  i  $y * z$ . Observem ara el següent:  $(x_0 * x_1 + y * z) - (x_0 * y + x_1 * z) = x_0(x_1 - y) + (y - x_1)z = x_0(x_1 - y) - (x_1 - y)z = (x_0 - z)(x_1 - y) > 0$ . L'últim pas és degut a que  $x_0 > z$  i  $x_1 > y$  ja que  $x_0$  i  $x_1$  són els elements majors de *S*, i són tots diferents. Per tant l'expressió original no era màxima ja que l'expressió resultant és major.

Anem a demostrar el resultat per inducció sobre *m*:

- *Cas base* ( $m = 0$ ). L'algorisme és correcte ja que retorna una expressió que suma zero i per tant és òptima.
- *Pas d'inducció*. Sigui  $m > 0$  i assumim la hipòtesi d'inducció: l'expressió màxima per un conjunt de  $< m$  elements es pot obtenir ordenant els elements de major a menor i agrupant-los de dos en dos consecutivament. Si ordenem els *m* elements  $x_0 > x_1 > x_2 > x_3 > \dots > x_{m-1}$ , sabem gràcies al resultat anterior que l'expressió òptima conté el producte  $x_0 * x_1$  seguit d'una expressió formada amb els nombres  $\{x_2, x_3, \dots, x_{m-1}\}$ . Aquesta expressió serà òbviament la major que puguem formar amb  $\{x_2, x_3, \dots, x_{m-1}\}$  i aplicant la hipòtesi d'inducció sabem que tindrà la forma  $x_2 * x_3 + \dots + x_{m-2} * x_{m-1}$ . Per tant, l'expressió òptima és  $x_0 * x_1 + x_2 * x_3 + \dots + x_{m-2} * x_{m-1}$ , com volíem demostrar.

**Proposta de solució al problema 4**

(a)

```
int f(const vector<int>& p, int l, int r){
 if (l + 1 ≥ r) return (p[l] ≤ p[r] ? l : r);
 else {
 int m = (l+r)/2;
 if (p[m] > p[m+1]) return f(p, m+1, r);
 else if (p[m-1] < p[m]) return f(p, l, m-1);
 else return m;
 }
}

pair<int,int> max_guany(const vector<int>& p) {
 return {f(p, 0, p.size()-1), p.size()-1};
}
```

El cost de *max\_guany* coincidirà amb el cost de la funció *f*. Per analitzar aquesta última, cal fixar-se que el seu cost ve donat per la recurrència  $T(n) = T(n/2) + \Theta(1)$ , d'on s'obté el cost de  $\Theta(\log n)$ .

(b)

```
int max_guany (const vector<int>& p, int k) {
 int m = p[k];
 for (int i = k - 1; i ≥ 0; --i)
 m = min(m, p[i]);

 int M = p[k];
 for (int i = k + 1; i < p.size (); ++i)
 M = max(M, p[i]);

 return M - m;
}
```

- (c) Podem utilitzar un algorisme de dividir i vèncer. Donat un vector *p*, el partim en dues meitats, separades pel punt mig *m*. Recursivament, calculem el màxim guany possible si comprem i venem a la part esquerra del vector, i a continuació, també recursivament, calculem el màxim guany possible si comprem i venem a la part dreta del vector. Finalment, utilitzant la funció de l'apartat anterior, calculem el màxim guany d'un període que inclou el punt mig *m* (és a dir, comprem a la part esquerra i venem a la part dreta). El resultat final és el màxim dels tres guanys calculats.

Hem desenvolupat un esquema de dividir i vèncer on fem dues crides recursives de mida la meitat, i a continuació fem un treball lineal per calcular el màxim guany que inclogui el punt *m*. Per tant, la recurrència que determina el cost de la funció és:  $T(n) = 2T(n/2) + \Theta(n)$ , que té com a solució asimptòtica  $\Theta(n \log n)$ .

*Nota:* hi ha solucions més eficients no basades en dividir i vèncer.

## Solució de l'Examen Parcial EDA

23/04/2020

## Proposta de solució al problema 2

Creuers, X35804:

```

#include <iostream>
#include <map>
using namespace std;

struct Info {
 string code;
 int price ;
};

int main() {
 map<int, Info> M;
 char c;
 while (cin >> c) {
 if (c == 'n') {
 cout << "num: " << M.size() << endl;
 }
 else if (c == 'u') {
 string code;
 int length, price;
 cin >> code >> length >> price;
 Info inf; inf.price = price; inf.code = code;
 M[length]={code,price};
 }
 else if (c == 'q') {
 int length;
 cin >> length;
 if (M.count(length) == 1) cout << M[length].price << endl;
 else cout << -1 << endl;
 }
 else if (c == 'p') {
 cout << string(10, '-') << endl;
 for (auto& p : M)
 cout << p.second.code << " " << p.first << " " << p.second.price << endl;
 cout << string(10, '*') << endl;
 }
 else {
 if (M.size() < 2) cout << "no" << endl;
 else {
 auto it = M.begin(); ++it;
 cout << it->second.code << " " << it->first << " " << it->second.price << endl;
 }
 }
 }
}

```

Donacions, X22314:

```

#include <iostream>
#include <map>
using namespace std;

int main() {
 map<string, int> M;
 char c;
 while (cin >> c) {
 if (c == 'N') {
 cout << "number: " << M.size() << endl;
 }
 else if (c == 'D') {
 string nif;
 int money;
 cin >> nif >> money;
 M[nif] += money;
 }
 else if (c == 'Q') {
 string nif;
 cin >> nif;
 if (M.count(nif) != 0) cout << M[nif] << endl;
 else cout << -1 << endl;
 }
 else if (c == 'P') {
 bool primer = true;
 for (auto& p : M) {
 if ((p.first[p.first.length()-2] - '0')%2 == 0) {
 cout << (primer?"":":") << p.first ;
 if (primer) primer = false;
 }
 }
 cout << endl;
 }
 else { // c == 'L'
 if (M.size() == 0) cout << "NO LAST NIF" << endl;
 else {
 auto it = M.end(); --it;
 cout << it->first << " " << it->second << endl;
 }
 }
 }
}

```



Efemèrides, X79163:

```
#include <iostream>
#include <map>
using namespace std;

struct Data {
 string event;
 int relevance;
};

int main() {
 int maximum_relevance = 0;
 map<string, Data> M;
 char c;
 while (cin >> c) {
 if (c == 'n') {
 cout << "number events: " << M.size() << endl;
 }
 else if (c == 's') {
 string date, event;
 int relevance;
 cin >> date >> event >> relevance;
 if (M.count(date)) cout << "ERROR: repeated date" << endl;
 else {
 M[date] = {event, relevance};
 if (relevance > maximum_relevance) maximum_relevance = relevance;
 }
 }
 else if (c == 'a') {
 string date;
 cin >> date;
 cout << M[date].event << endl;
 }
 else if (c == 'm') {
 cout << "maximum relevance: " << maximum_relevance << endl;
 }
 else { // c == 'e'
 if (M.size() < 2) cout << "ERROR: at least two events needed" << endl;
 else {
 int total = 0;
 auto it = M.end();
 --it;
 total += it->second.relevance;
 it = M.begin();
 total += it->second.relevance;
 cout << total << endl;
 }
 }
 }
}
```

```

 }
 }
}

```

### Proposta de solució al problema 3

Vector xulo, X30043:

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int position (const vector<int>& v, int e, int d) {
 if (e+1 == d) return e;
 int m = (e+d)/2;
 if (v[e] ≥ v[m]) return position (v, m, d);
 else return position (v, e, m);
}

int search (int x, const vector<int>& v, int e, int d) {
 if (e > d) return -1;
 if (e == d) return (v[e] == x ? e : -1);
 int m = (e + d)/2;
 if (x < v[m]) return search (x, v, m + 1, d);
 else return search (x, v, e, m);
}

int search (int x, const vector<int>& v) {
 int n = v.size ();
 int j = position (v, 0, n-1);
 int p = search (x, v, 0, j);
 if (p ≠ -1) return p;
 return search (x, v, j+1, n-1);
}

```

Vector xulo, X74873:

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int position (const vector<int>& v, int e, int d) {
 if (e+1 == d) return e;
 int m = (e+d)/2;
 if (v[e] ≥ v[m]) return position (v, m, d);
 else return position (v, e, m);
}

int search (int x, const vector<int>& v, int e, int d) {
 if (e > d) return -1;

```

```

 if (e == d) return (v[e] == x ? e : -1);
 int m = (e + d + 1)/2;
 if (x > v[m]) return search(x, v, e, m - 1);
 else return search(x, v, m, d);
}

```

```

int search(int x, const vector<int>& v) {
 int n = v.size ();
 int j = position(v, 0, n-1);
 int p = search(x, v, 0, j);
 if (p != -1) return p;
 return search(x, v, j+1, n-1);
}

```

Vector xulo, X83303:

```

#include <iostream>
#include <vector>
using namespace std;

int position(const vector<int>& v, int e, int d) {
 if (e+1 == d) return e;
 int m = (e+d)/2;
 if (v[e] ≤ v[m]) return position(v, m, d);
 else return position(v, e, m);
}

int search(int x, const vector<int>& v, int e, int d) {
 if (e > d) return -1;
 if (e == d) return (v[e] == x ? e : -1);
 int m = (e + d)/2;
 if (x > v[m]) return search(x, v, m + 1, d);
 else return search(x, v, e, m);
}

int search(int x, const vector<int>& v) {
 int n = v.size ();
 int j = position(v, 0, n-1);
 int p = search(x, v, 0, j);
 if (p != -1) return p;
 return search(x, v, j+1, n-1);
}

```

Vector xulo, X90362:

```

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int position(const vector<int>& v, int e, int d) {
 if (e+1 == d) return e;

```

```
 int m = (e+d)/2;
 if (v[e] ≤ v[m]) return position (v, m, d);
 else return position (v, e, m);
}

int search(int x, const vector<int>& v, int e, int d) {
 if (e > d) return -1;
 if (e == d) return (v[e] == x ? e : -1);
 int m = (e + d + 1)/2;
 if (x < v[m]) return search(x, v, e, m - 1);
 else return search(x, v, m, d);
}

int search(int x, const vector<int>& v) {
 int n = v.size ();
 int j = position (v, 0, n-1);
 int p = search(x, v, 0, j);
 if (p ≠ -1) return p;
 return search(x, v, j+1, n-1);
}
```

## Solució de l'Examen Parcial EDA

06/11/2020

## Proposta de solució al problema 1

```

(a) bool tri_search (const vector<int>& v, int l, int r, int x) {
 if (l > r) return false;
 else {
 int n_elems = (r-l+1);
 int f = l + n_elems/3;
 int s = r - n_elems/3;
 if (v[f] == x or v[s] == x) return true;
 if (x < v[f]) return tri_search (v, l, f-1, x);
 if (x < v[s]) return tri_search (v, f+1, s-1, x);
 else return tri_search (v, s+1, r, x);
 }
}

```

En el cas pitjor es fan totes les crides recursives fins que  $l > r$ . La recurrència que expressa el cost del programa en aquest cas és:

$$T(n) = T(n/3) + \Theta(1)$$

que té solució  $T(n) \in \Theta(\log n)$ .

(b) Siguin  $f = n(\log n)^{1/2}$  i  $g = n(\log n)^{1/3}$ .

Per veure que són  $\Omega(n)$  i no  $\Theta(n)$  només cal veure que els límits següents són infinit:

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/2}}{n} = \lim_{x \rightarrow \infty} (\log n)^{1/2} = \infty$$

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/3}}{n} = \lim_{x \rightarrow \infty} (\log n)^{1/3} = \infty$$

Per veure que són  $O(n \log n)$  i no  $\Theta(n \log n)$  només cal veure que els límits següents són zero:

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/2}}{n \log n} = \lim_{x \rightarrow \infty} \frac{(\log n)^{1/2}}{\log n} = \lim_{x \rightarrow \infty} \frac{1}{(\log n)^{1/2}} = 0$$

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/3}}{n \log n} = \lim_{x \rightarrow \infty} \frac{(\log n)^{1/3}}{\log n} = \lim_{x \rightarrow \infty} \frac{1}{(\log n)^{2/3}} = 0$$

Finalment per veure que  $f \notin \Theta(g)$ , vegem que el límit següent no és una constant major estricta que zero:

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^{1/3}}{n(\log n)^{1/2}} = \lim_{x \rightarrow \infty} \frac{(\log n)^{1/3}}{(\log n)^{1/2}} = \lim_{x \rightarrow \infty} \frac{1}{(\log n)^{1/6}} = 0$$

### Proposta de solució al problema 2

- (a) La idea d'aquest algorisme és que, cada vegada que trobem un natural es compten les aparicions posteriors d'aquest en el vector i es marquen amb un  $-1$  per a no considerar-les més en el futur. Per tant, quan visitem un element marcat amb un  $-1$  ens estalviem el bucle més intern.

Si construïm un vector on tots els nombres són diferents, aleshores aquesta optimització no serveix per a res. A més, si tots els nombres són diferents no tenim cap element dominant (a no ser que  $n = 1$ ) i els dos bucles s'executen el màxim nombre de vegades. El cos del bucle més intern és clarament constant, pel que només hem de comptar quantes vegades s'executa. Donat una  $i$  concreta, el bucle intern s'executa  $n - i$  vegades. Com que  $i$  va des de 0 fins a  $n - 1$ , el cost total és  $n + (n - 1) + (n - 2) + \dots + 1 = \Theta(n^2)$ .

El cost en cas millor es dona, per exemple, quan tenim un vector amb un únic element repetit  $n$  vegades. En aquest cas, quan  $i = 0$  visitarem tots els elements del vector marcant-los amb un  $-1$ . Per a totes les altres  $i$ , el bucle més intern no s'executarà. Per tant, el cost en cas millor és  $\Theta(n)$ .

Si ens asseguren que tenim com a molt 100 naturals diferents, aleshores el bucle intern s'executarà com a molt 100 vegades. És a dir, hi haurà com a molt 100  $i$ s per les quals el bucle intern s'executarà. Aquestes  $i$ s contribuiran en el cas pitjor un cost de  $\Theta(100n) = \Theta(n)$ . Per la resta de les  $i$ s (en tenim com a màxim  $n$ ) el bucle intern no s'executarà i per tant, contribuiran amb un cost de  $\Theta(n)$ . Així doncs, el cost en cas pitjor ha canviat i ha passat a ser  $\Theta(n)$ .

- (b) Per aquest exercici primer recordem que la ordenació per inserció té cost  $\Theta(n^2)$  en cas pitjor i  $\Theta(n)$  en cas millor. Pel que fa al *quicksort*, tenim un cost de  $\Theta(n^2)$  en cas pitjor i  $\Theta(n \log n)$  en cas millor. Per analitzar el cost de *dominant\_sort*, oblidem-nos de moment de la crida a *own\_sort*. La resta del bucle veiem que com a màxim visita cada element del vector una vegada, fent-hi un treball constant. Cal remarcar que a vegades no visita tots els elements, ja que s'atura quan detecta l'element dominant. El cas pitjor el tenim quan visita tots els elements i no troba cap dominant (això triga  $\Theta(n)$ ). El cas millor es dona quan el primer element que visita és el dominant, però podem observar que per a detectar que és dominant ha de visitar almenys  $n/2$  elements, pel que el cost també és  $\Theta(n)$ . Per tant, el codi sempre triga  $\Theta(n)$  (obviant la crida a *own\_sort*).

Si *own\_sort* és una ordenació per inserció, el cost en cas millor és  $\Theta(n) + \Theta(n) = \Theta(n)$ , i en cas pitjor és  $\Theta(n) + \Theta(n^2) = \Theta(n^2)$ .

Si *own\_sort* és un *quicksort*, el cost en cas millor és  $\Theta(n) + \Theta(n \log n) = \Theta(n \log n)$ , i en cas pitjor és  $\Theta(n) + \Theta(n^2) = \Theta(n^2)$ .

- (c) El codi complet és:

```
int dominant_divide (const vector<int>& v, int l, int r) {
 if (l == r) return v[l];
 int n_elems = (r-l+1);
 int m = (l+r)/2;
 int maj_left = dominant_divide(v, l, m);
 if (maj_left != -1 and times(v, l, r, maj_left) > n_elems/2) return maj_left;
 int maj_right = dominant_divide(v, m+1, r);
 if (maj_right != -1 and times(v, l, r, maj_right) > n_elems/2) return maj_right;
 return -1; }
```

Per analitzar el seu cost ens adonem que en el cas pitjor es fan dues crides recursives de tamany la meitat i dues crides a *times*. La resta del codi té cost constant. Observem ara que la funció *times* rep *v* per referència i per tant el seu cost es pot descriure per  $T(n) = T(n - 1) + \Theta(1)$ , que té solució  $T(n) \in \Theta(n)$ . Així doncs, la recurrència que descriu els cost en cas pitjor d'aquest programa és

$$T(n) = 2T(n/2) + \Theta(n)$$

que té solució  $T(n) \in \Theta(n \log n)$ .

**Solució de l'Examen Parcial EDA****15/04/2021****Proposta de solució al problema 1**

(a) Calculem primer el límit

$$\lim_{x \rightarrow \infty} \frac{2^{2n}}{2^n} = \lim_{x \rightarrow \infty} 2^n = \infty$$

.

Per tant, l'única afirmació certa és que  $2^{2n} \in \Omega(2^n)$ .

A continuació, calculem el límit

$$\lim_{x \rightarrow \infty} \frac{\log(2n)}{\log(n)} = \lim_{x \rightarrow \infty} \frac{\log(2) + \log(n)}{\log(n)} = \lim_{x \rightarrow \infty} \frac{\log(2)}{\log(n)} + \lim_{x \rightarrow \infty} \frac{\log(n)}{\log(n)} = 0 + 1 = 1$$

.

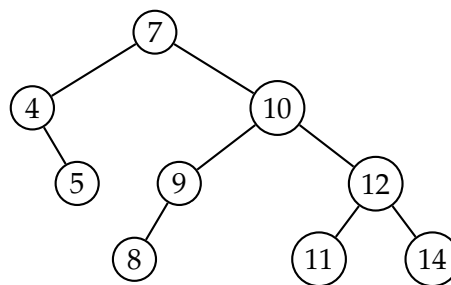
Així doncs,  $\log(2n) \in \Theta(\log(n))$ . Per tant també  $\log(2n) \in O(\log(n))$  i  $\log(2n) \in \Omega(\log(n))$ .

(b) El cos de cada iteració del bucle és  $\Theta(1)$  i per tant només cal comptar quantes iteracions es fan. Si  $y_t$  denota el valor de  $y$  al final de la  $t$ -èsima iteració, el que busquem és el mínim  $t \geq 0$  tal que  $y_t > n$ . Sabem que  $y_t = 1 + 2 + 3 + 4 + \dots + (t + 1) = \Theta(t^2)$  i per tant el mínim  $t$  és  $\Theta(\sqrt{n})$ .

(c) El codi llegeix dues seqüències d' $n$  enters i calcula si les dues seqüències tenen intersecció no buida.

El cas pitjor es dona quan tots els elements de la primera seqüència són diferents. Això farà que hi hagi  $n$  elements al *map*. Sabem que en C++ buscar un element en un *map* té cost logarítmic en el nombre d'elements. Per tant, en el cas pitjor es donen  $n$  voltes al bucle, i cada volta té cost logarítmic, pel que el cost total en cas pitjor és  $\Theta(n \log n)$ .

(d) L'arbre AVL resultant és:

**Proposta de solució al problema 2**(a) Ho demostrarem per inducció sobre  $n$ .

*Cas base:* ( $n = 0$ ). En aquest cas, la graella té mida  $1 \times 1$  i per tant només té una casella, que ha de ser necessàriament la casella bloquejada. Així doncs, no hi ha caselles restants per omplir i el resultat es compleix trivialment.



*Pas d'inducció:* sigui  $n > 0$  i assumim que el resultat és cert per graelles de mida  $2^{n-1} \times 2^{n-1}$ . Aleshores podem partir la graella en 4 parts iguals, que seran subgraelles de mida  $2^{n-1} \times 2^{n-1}$ . Si, tal com es mostra a la figura de l'enunciat, la casella bloquejada cau a la subgraella de baix a l'esquerra, aleshores podem col·locar una peça a la part central de manera que bloqueja exactament una casella a les altres 3 subgraelles. Si la casella bloquejada cau a una altra subgraella, sempre podrem escollir una peça que bloquegi exactament una casella a les altres 3 subgraelles. Per tant, després d'haver col·locat aquest peça central sempre tindrem 4 subgraelles de mida  $2^{n-1} \times 2^{n-1}$  amb exactament una casella bloquejada a cadascuna d'elles. Gràcies a la hipòtesi d'inducció sabem que totes elles es poden omplir amb les peces disponibles, i per tant la graella original de mida  $2^n \times 2^n$  també.

(b)

```

int quadrant(Coord pos, int i_l, int i_r, int j_l, int j_r) {
 int size = j_r - j_l + 1;
 int i_m = i_l + size / 2;
 int j_m = j_l + size / 2;
 if (pos.first < i_m and pos.second < j_m) return 0;
 if (pos.first < i_m) return 1;
 if (pos.second < j_m) return 2;
 return 3;
}

void fill (vector<vector<int>>& M, int i_l, int i_r, int j_l, int j_r,
 Coord c_blocked, int& num){
 if (i_l == i_r) return; // 1x1
 int size = j_r - j_l + 1;
 int i_m = i_l + size / 2; // Midpoints
 int j_m = j_l + size / 2;

 vector<Coord> coords_blocked(4); // Blocked cell in each quadrant
 coords_blocked [0] = {i_m - 1, j_m - 1};
 coords_blocked [1] = {i_m - 1, j_m};
 coords_blocked [2] = {i_m, j_m - 1};
 coords_blocked [3] = {i_m, j_m};

 int q = quadrant(c_blocked, i_l, i_r, j_l, j_r);
 coords_blocked [q] = c_blocked;

 for (int k = 0; k < 4; ++k)
 if (M[coords_blocked[k].first][coords_blocked[k].second] == -1)
 M[coords_blocked[k].first][coords_blocked[k].second] = num;

 ++num;

 fill (M, i_l, i_m - 1, j_l, j_m - 1, coords_blocked [0], num); // Q0
 fill (M, i_l, i_m - 1, j_m, j_r, coords_blocked [1], num); // Q1
 fill (M, i_m, i_r, j_l, j_m - 1, coords_blocked [2], num); // Q2
 fill (M, i_m, i_r, j_m, j_r, coords_blocked [3], num); // Q3
}

```

- (c) Per analitzar el cost del codi, ens hem de fixar en la funció *fill*. Fixem-nos que el seu codi essencialment no té bucles. Només n'hi ha un, però sempre dóna 4 voltes (independentment de la mida del problema). Per tant, fa un nombre constant de voltes. Com que totes les altres instruccions triguen temps  $\Theta(1)$ , si no consideréssim les crides recursives la funció trigaria temps constant. No obstant, per un problema de mida  $2^n \times 2^n$  es fan 4 crides recursives amb mida  $2^{n-1} \times 2^{n-1}$ . Per tant, el cost en funció de  $n$  ve donat per la recurrència:

$$T(n) = 4T(n-1) + \Theta(1)$$

que té solució  $T(n) \in \Theta(4^n)$ .

Com que el nombre de caselles és  $2^n \cdot 2^n = 4^n$ , podem afirmar que el codi és lineal en el nombre de caselles.

## Solució de l'Examen Parcial EDA

08/11/2021

## Proposta de solució al problema 1

- (a) La funció
- mystery*
- determina si
- $n$
- és un nombre primer.

Pel que fa al seu cost, fixem-nos que les tres primeres línies del codi tenen cost constant perquè només fan operacions aritmètiques i comparacions entre enters. La part interessant del codi és el bucle. En el cas pitjor es fan totes les voltes al bucle. Per facilitar el raonament podem assumir que el bucle comença a  $i = 1$  (afegir dues voltes al bucle no canvia el cost asimptòtic). Com que parem quan  $i^2 > n$ , és a dir  $i > \sqrt{n}$ , es farien  $\sqrt{n}$  voltes si  $i$  s'incrementés en una unitat a cada volta. Com que s'incrementa en dos, se'n fan la meitat:  $\sqrt{n}/2$  voltes. Si considerem que cada volta només fa un treball constant (assignacions, operacions aritmètiques i comparacions entre enters), el cost total en cas pitjor és  $\Theta(\sqrt{n})$ .

- (b) El seu cost és
- $\Theta(n\sqrt{n}) = \Theta(n^{3/2})$
- .

Les dues primeres línies tenen cost constant. Anem a estudiar el bucle. El primer fet a observar és que el bucle més intern (el que varia  $k$ ) té cost  $\Theta(n)$ . El bucle més extern fa  $n$  voltes. Per algunes d'elles s'executarà el bucle intern i per les altres es farà un treball constant. Anem a comptar quantes vegades s'executa el bucle intern. Aquest només s'executa quan  $i = j^2$ . La variable  $j$  inicialment val zero, i cada vegada que s'executa el bucle intern s'incrementa en una unitat. Ens podem fixar que el bucle intern s'executarà per primera vegada quan  $i = j^2 = 0$ , i llavors  $j$  passarà a valer 1. La segona vegada serà quan  $i = j^2 = 1$  i llavors  $j$  passarà a valer 2. La tercera vegada serà quan  $i = j^2 = 4$ , i així successivament. Per tant, el bucle intern s'executarà tantes vegades com quadrats hi hagi entre 0 i  $n - 1$ , és a dir,  $\lceil \sqrt{n} \rceil$ . La resta de vegades, és a dir,  $(n - \lceil \sqrt{n} \rceil)$  vegades, el bucle intern no s'executa. Així doncs el cost total és  $(n - \lceil \sqrt{n} \rceil) \cdot \Theta(1) + \lceil \sqrt{n} \rceil \cdot \Theta(n) = \Theta(n \cdot \sqrt{n})$ .

- (c) Ens fixem que *pairs* és una funció recursiva on a cada crida recursiva decreix el nombre d'elements en  $v[l \dots r]$ . Inicialment aquest nombre és  $n$ . Si ens centrem en el cas recursiu, podem veure que hi ha dues crides recursives on el nombre d'elements és la meitat, una sèrie d'instruccions de cost constant i dos bucles enniestrats. El bucle extern tracta la part esquerra del vector, entre  $l$  i  $m$ , on hi ha essencialment  $n/2$  elements. Per cadascun d'aquests elements, el bucle intern recorre la part dreta del vector, entre  $m + 1$  i  $r$ , on també tenim aproximadament  $n/2$  elements. Tot plegat, el condicional de dins el bucle s'executa bàsicament  $n^2/4$  vegades, que equival a un treball  $\Theta(n^2)$ . Per tant, la recurrència que descriu el cost d'aquesta funció és

$$T(n) = 2 \cdot T(n/2) + \Theta(n^2)$$

que té solució  $T(n) \in \Theta(n^2)$ .

- (d) Com que  $K$  no depèn d' $n$ , la primera línia del codi té cost constant, així com la segona. El primer bucle té cost  $\Theta(n)$ . Només ens falta analitzar el segon bucle, que repeteix  $K$  vegades un treball constant. Com que  $K$  també és constant, el bucle triga  $\Theta(1)$ . Tot plegat el cost de la funció és  $\Theta(n)$ .

La correcció del codi es basa en la següent observació. El primer bucle calcula, per cada nombre  $k$  entre 0 i  $K - 1$ , quantes vegades apareix  $k$  a  $v$ . Aquest valor es guarda a *times*[ $k$ ]. Si un nombre  $k$  apareix  $p$  vegades, aleshores hi haurà exactament  $p \cdot (p - 1)/2$

parelles  $(i, j)$  amb  $0 \leq i < j < n$  tals que  $v[i] = v[j] = k$ . Com que suma sobre totes les  $k$  possibles, el segon bucle ens calcula la quantitat desitjada.

### Proposta de solució al problema 2

```
(a) int first_occurrence (int x, const vector<int>& v, int l, int r) {
 if (l > r) return -1;
 else {
 int m = (l+r)/2;
 if (v[m] < x) return first_occurrence (x, v, m+1, r);
 else if (v[m] > x) return first_occurrence (x, v, l, m-1);
 else if (m == l or v[m-1] != x) return m;
 else return first_occurrence (x, v, l, m-1);
 }
 }
```

(b) El codi a omplir és  $res = n - q - p$ ;

Per analitzar el seu cost, veiem que el codi comença amb una crida a *first\_occurrence* de cost, en cas pitjor,  $O(\log n)$ . A continuació la propera part interessant és el primer bucle, de cost  $\Theta(n)$ . El segon bucle, tot i que només visita la meitat dels elements de  $v$ , té el mateix cost  $\Theta(n)$ , perquè sabem que un *swap* triga temps  $\Theta(1)$ . Finalment, tenim una altra crida a *first\_occurrence*, altra vegada de cost, en cas pitjor,  $O(\log n)$ . Tot plegat el cost del codi és  $\Theta(n)$ .

Per tal d'entendre per què el codi és correcte, hem d'entendre que essencialment aquest intenta calcular la primera i la darrera aparició d' $x$  a  $v$ , i calcular quants elements hi ha entre aquestes dues posicions. El punt clau és adonar-se que si invertim el vector i neguem tots els seus elements, obtenim un vector ordenat creixentment tal que l'última aparició d' $x$  en el  $v$  original coincideix amb la primera aparició de  $-x$  en el nou  $v$ . A més, si la primera aparició de  $-x$  en el nou vector  $v$  és a la posició  $q$ , aleshores la darrera aparició d' $x$  en el vector  $v$  original és  $n - 1 - q$ . Per tant, el nombre d'elements entre  $p$  (primera aparició) i  $n - 1 - q$  (última aparició) és  $(n - 1 - q) - p + 1 = n - q - p$ .

(c) Podem aconseguir fàcilment un algorisme de cost, en cas pitjor,  $O(\log n)$  si calculem millor la darrera aparició d' $x$  a  $v$ . Per tal de fer-ho, podem modificar el codi de *first\_occurrence* lleugerament:

```
int last_occurrence (int x, const vector<int>& v, int l, int r) {
 if (l > r) return -1;
 else {
 int m = (l+r)/2;
 if (v[m] < x) return last_occurrence (x, v, m+1, r);
 else if (v[m] > x) return last_occurrence (x, v, l, m-1);
 else if (m == r or v[m+1] != x) return m;
 else return last_occurrence (x, v, m+1, r);
 }
}
```

Una crida a *last\_occurrence* té cost en cas pitjor  $O(\log n)$ . Aleshores, només ens caldrà trobar la primera i la darrera aparició d' $x$ , diguem-ne  $p$  i  $q$ , i retornar  $q - p + 1$  o bé 0 si no hi ha cap aparició.

---

## Solucions d'Exàmens d'Ordinador

## Solució de l'Examen d'Ordinador EDA

13/12/2010

Torn 1

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>
#include <queue>
#include <cassert>

using namespace std;

const int N_DIRS = 4;
const int di[N_DIRS] = { 1, 0, -1, 0};
const int dj[N_DIRS] = { 0, 1, 0, -1};

struct Pos {
 int i, j;
 Pos(int ii = -1, int jj = -1): i(ii), j(jj) {}
};

bool ok(int n, int m, int i, int j) {
 return
 0 ≤ i and i < n and
 0 ≤ j and j < m;
}

int search(const vector< vector<char> >& map, int n, int m, int i0, int j0) {
 const int MINFTY = -1;
 vector< vector<int> > dist(n, vector<int>(m, MINFTY));
 queue<Pos> q;
 int max_dist = MINFTY;
 q.push(Pos(i0, j0));
 dist[i0][j0] = 0;
 while (not q.empty()) {
 Pos p = q.front();
 q.pop();
 int i = p.i;
 int j = p.j;
 for(int k = 0; k < N_DIRS; ++k) {
 int ii = i + di[k];
 int jj = j + dj[k];
 if (ok(n, m, ii, jj) and map[ii][jj] ≠ 'X' and dist[ii][jj] == MINFTY) {
 q.push(Pos(ii, jj));
 dist[ii][jj] = 1 + dist[i][j];
 if (map[ii][jj] == 't') max_dist = dist[ii][jj];
 }
 }
 }
}

```

```

 return max_dist;
}

int main(void) {
 int n, m;
 cin >> n >> m;
 vector < vector<char> > map(n, vector<char>(m));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j)
 cin >> map[i][j];
 int f, c;
 cin >> f >> c;
 --f;
 --c;
 int dist = search(map, n, m, f, c);
 if (dist ≥ 0) cout << "distancia maxima: " << dist << endl;
 else cout << "no es pot arribar a cap tresor" << endl;
}

```

### Proposta de solució al problema 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<bool> Vec;
typedef vector<Vec > Mat;

void next(int i, int j, int n, int& ni, int& nj) {
 if (j < n-1) {
 ni = i;
 nj = j+1;
 }
 else {
 ni = i+1;
 nj = 0;
 }
}

const int N_DIRS = 8;
const int DI[N_DIRS] = { -1, -1, 0, 1, 1, 1, 0, -1};
const int DJ[N_DIRS] = { 0, -1, -1, -1, 0, 1, 1, 1};

bool ok(int i, int j, int n) {
 return
 0 ≤ i and i < n and
 0 ≤ j and j < n;
}

```

```

bool safe(int i, int j, int n, Mat& m) {
 for (int k = 0; k < 8; ++k) {
 int ii = i + DI[k];
 int jj = j + DJ[k];
 if (ok(ii, jj, n) and m[ii][jj]) return false;
 }
 return true;
}

void escriu(int n, const Mat& m) {
 for (int i = 0; i < n; ++i) {
 for (int j = 0; j < n; ++j)
 if (m[i][j]) cout << 'K';
 else cout << '.';
 cout << endl;
 }
 cout << "-----" << endl;
}

void search(int i, int j, int n, int r, int s, Mat& m) {
 if (s == r) escriu(n, m);
 else if (i != n) {
 int ni, nj;
 next(i, j, n, ni, nj);
 if (safe(i, j, n, m)) {
 m[i][j] = true;
 search(ni, nj, n, r, s+1, m);
 }
 m[i][j] = false;
 search(ni, nj, n, r, s, m);
 }
}

int main(void) {
 int n, r;
 cin >> n >> r;
 Mat m(n, Vec(n, false));
 search(0, 0, n, r, 0, m);
}

```



**Torn 2****Proposta de solució al problema 1**

```

#include <iostream>
#include <string>
#include <map>

using namespace std;

typedef map<string, int>::iterator Iterator ;

int main(void) {
 map<string, int> bagmul;
 string command;
 while (cin >> command) {
 if (command == "store") {
 string word;
 cin >> word;
 pair<Iterator , bool> res = bagmul.insert (pair<string,int>(word, 1));
 if (not res.second) ++bagmul[word];
 }
 else if (command == "delete") {
 string word;
 cin >> word;
 Iterator i = bagmul.find(word);
 if (i != bagmul.end()) {
 if (i->second == 1) bagmul.erase(i);
 else --bagmul[word];
 }
 }
 else if (command == "minimum?") {
 if (bagmul.empty()) cout << "indefinite minimum" << endl;
 else {
 Iterator i = bagmul.begin ();
 cout << "minimum: "
 << i->first << ", "
 << i->second << " time(s)" << endl;
 }
 }
 else {
 if (bagmul.empty()) cout << "indefinite maximum" << endl;
 else {
 Iterator i = bagmul.end();
 --i;
 cout << "maximum: "
 << i->first << ", "
 << i->second << " time(s)" << endl;
 }
 }
 }
}

```

```

 }
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> CVec;
typedef vector<CVec> CMat;
typedef vector<bool> BVec;
typedef vector<BVec> BMat;

const int N_DIRS = 4;
const int DR[N_DIRS] = { 0, -1, 0, 1 };
const int DC[N_DIRS] = { 1, 0, -1, 0 };

struct Point {
 int r, c;
 Point(int rr, int cc) : r(rr), c(cc) { }
};

bool ok(int r, int c, int n, int m) {
 return
 0 ≤ r and r < n and
 0 ≤ c and c < m;
}

void search(int re, int ce, int n, int m, const CMat& t, vector<Point>& v, BMat& seen) {
 Point p = v.back ();
 if (p.r == re and p.c == ce) {
 for (int i = 0; i < v.size (); ++i)
 cout << t[v[i].r][v[i].c];
 cout << endl;
 }
 else {
 for (int k = 0; k < N_DIRS; ++k) {
 int rr = p.r + DR[k];
 int cc = p.c + DC[k];
 if (ok(rr, cc, n, m) and not seen[rr][cc]) {
 seen[rr][cc] = true;
 v.push_back(Point(rr, cc));
 search(re, ce, n, m, t, v, seen);
 v.pop_back ();
 seen[rr][cc] = false;
 }
 }
 }
}

```

```
 }
}

int main(void) {
 int n, m;
 cin >> n >> m;
 CMat t(n, CVec(m));
 BMat seen(n, BVec(m, false));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j)
 cin >> t[i][j];
 int ri, ci, re, ce;
 cin >> ri >> ci >> re >> ce;
 seen[ri][ci] = true;
 vector<Point> v(1, Point(ri, ci));
 search(re, ce, n, m, t, v, seen);
}
```

## Solució de l'Examen d'Ordinador EDA

19/5/2011

## Proposta de solució al problema 1

(5 punts)

```

#include <iostream>
#include <vector>

using namespace std;

struct Point {
 int r, c;
 Point(int rr, int cc) : r(rr), c(cc) {}
};

bool ok(int n, int m, const Point& p) {
 return
 0 ≤ p.r and p.r < n and
 0 ≤ p.c and p.c < m;
}

const int N_DIRS_KNIGHT = 8;
const int DR_KNIGHT[8] = {-2, -1, 2, 1, -2, -1, 2, 1};
const int DC_KNIGHT[8] = {-1, -2, -1, -2, 1, 2, 1, 2};

const int N_DIRS_BISHOP = 4;
const int DR_BISHOP[4] = {1, -1, 1, -1};
const int DC_BISHOP[4] = {1, 1, -1, -1};

int dfs(int n, int m, const Point& p,
 vector< vector<char> >& map, vector< vector<bool> >& marked,
 const int N_DIRS, const int DR[], const int DC[]) {
 int s = 0;
 marked[p.r][p.c] = true;
 for (int i = 0; i < N_DIRS; ++i) {
 Point q(p.r + DR[i], p.c + DC[i]);
 if (ok(n,m,q) and map[q.r][q.c] ≠ 'T' and not marked[q.r][q.c])
 s += dfs(n, m, q, map, marked, N_DIRS, DR, DC);
 }
 if ('0' ≤ map[p.r][p.c] and map[p.r][p.c] ≤ '9') {
 s += map[p.r][p.c] - '0';
 map[p.r][p.c] = '.';
 }
 return s;
}

int main(void) {
 int n, m;
 while (cin >> n >> m) {
 vector<Point> knights, bishops;

```

```

vector< vector<char> > map(n, vector<char>(m));
for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j) {
 cin >> map[i][j];
 switch(map[i][j]) {
 case 'K': knights.push_back(Point(i,j)); break;
 case 'B': bishops.push_back(Point(i,j)); break;
 };
 }
int s = 0;
vector< vector<bool> > marked_knight(n, vector<bool>(m, false));
for (int k = 0; k < knights.size (); ++k) {
 Point p = knights[k];
 if (not marked_knight[p.r][p.c])
 s += dfs(n, m, p, map, marked_knight, N_DIRS_KNIGHT, DR_KNIGHT, DC_KNIGHT);
}
vector< vector<bool> > marked_bishop(n, vector<bool>(m, false));
for (int k = 0; k < bishops.size (); ++k) {
 Point p = bishops[k];
 if (not marked_bishop[p.r][p.c])
 s += dfs(n, m, p, map, marked_bishop, N_DIRS_BISHOP, DR_BISHOP, DC_BISHOP);
}
cout << s << endl;
}
}

```

## Proposta de solució al problema 2

(5 punts)

```

#include <iostream>
#include <set>

using namespace std;

typedef long long int lint ;

int main(void) {

 lint suma = 0;
 set<lint> selec ;
 set<lint> resta ;

 int n;
 cin >> n;
 string op;
 lint val;
 while (cin >> op >> val) {
 if (op == "deixar") {
 if (selec.size () < n) {

```

```

 selec . insert (val);
 suma += val;
 }
 else {
 lint min = *(selec . begin ());
 if (min < val) {
 selec . insert (val);
 selec . erase (selec . begin ());
 suma = suma + val - min;
 resta . insert (min);
 }
 else resta . insert (val);
 }
}
else {
 if (*(selec . begin ()) ≤ val) {
 selec . erase (val);
 suma -= val;
 if (resta . size () > 0) {
 set<lint>:: iterator it = resta . end ();
 --it;
 selec . insert (* it);
 suma += *it;
 resta . erase (it);
 }
 }
 else resta . erase (val);
}
cout << suma << endl;
}
}

```

## Solució de l'Examen d'Ordinador EDA

13/12/2011

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> VC;
typedef vector<VC> MC;
typedef vector<bool> VB;
typedef vector<VB> MB;

const int N_DIRS = 4;
const int DI[N_DIRS] = { 1, 0, -1, 0};
const int DJ[N_DIRS] = { 0, 1, 0, -1};

bool ok(int n, int m, int i, int j) {
 return
 0 ≤ i and i < n and
 0 ≤ j and j < m;
}

int search(const MC& map, int n, int m, int i, int j, MB& marked) {
 int t = 0;
 if (map[i][j] == 't') ++t;
 marked[i][j] = true;
 for (int k = 0; k < N_DIRS; ++k) {
 int ii = i + DI[k];
 int jj = j + DJ[k];
 if (ok(n, m, ii, jj) and map[ii][jj] ≠ 'X' and not marked[ii][jj])
 t += search(map, n, m, ii, jj, marked);
 }
 return t;
}

int main(void) {
 int n, m;
 cin >> n >> m;
 MC map(n, VC(m));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j)
 cin >> map[i][j];
 int f, c;
 cin >> f >> c;

 MB marked(n, VB(m, false));
 cout << search(map, n, m, f-1, c-1, marked) << endl;
}

```

**Proposta de solució al problema 2**

```
#include <iostream>
#include <vector>

using namespace std;

void b(int k, int n, string& s) {
 if (k == n) cout << s << endl;
 else {
 s[k] = 'A'; b(k+1, n, s);
 s[k] = 'C'; b(k+1, n, s);
 s[k] = 'G'; b(k+1, n, s);
 s[k] = 'T'; b(k+1, n, s);
 }
}

int main() {
 int n;
 cin >> n;
 string s(n, 'X');
 b(0, n, s);
}
```



## Solució de l'Examen d'Ordinador EDA

14/05/2012

## Proposta de solució al problema 1

(5 punts)

```

#include<iostream>
#include<string>
#include<map>

using namespace std;

typedef map<string,string>::iterator ite ;

int main() {
 string s;
 map<string,string> liats;
 while(cin >> s) {
 if (s == "info") {
 cout << "PARELLES:" << endl;
 for (ite it = liats.begin(); it != liats.end(); ++it)
 if (it->second != "" and it->first < it->second)
 cout << it->first << " " << it->second << endl;
 cout << "SOLS:" << endl;
 for (ite it = liats.begin(); it != liats.end(); ++it)
 if (it->second == "")
 cout << it->first << endl;
 cout << "-----" << endl;
 }
 else {
 string x, y;
 cin >> x >> y;
 if (liats[x] != "") liats[liats[x]] = "";
 if (liats[y] != "") liats[liats[y]] = "";
 liats[x] = y;
 liats[y] = x;
 }
 }
}

```

## Proposta de solució al problema 2

(5 punts)

```

#include<iostream>
#include<vector>
#include<queue>
#include<algorithm>

using namespace std;

const int xf[8] = {-1,-1,-1,0,0,1,1,1};

```

```

const int yf[8] = {-1,0,1,-1,1,-1,0,1};
const int x[4] = {0,0,1,-1};
const int y[4] = {1,-1,0,0};

bool cerca_bolet (vector<vector<char> >& M, pair<int,int> p) {
 queue<pair<int, int> > Q;
 if (M[p.first][p.second] == 'X') return false;
 M[p.first][p.second] = 'X'; Q.push(p);
 while (not Q.empty()) {
 pair<int,int> q = Q.front (); Q.pop();
 for (int i = 0; i < 4; ++i) {
 int u = q.first + x[i];
 int v = q.second + y[i];
 if (M[u][v] == 'B') return true;
 if (M[u][v] != 'X') {
 M[u][v] = 'X';
 Q.push(make_pair(u,v));
 }
 }
 }
 return false;
}

int main() {
 int f, c;
 while (cin >> f >> c) {
 vector<vector<char> > M(f, vector<char> (c));
 pair<int, int> p;
 queue<pair<int,int> > F;
 for (int i = 0; i < f; ++i) {
 for (int j = 0; j < c; ++j) {
 cin >> M[i][j];
 if (M[i][j] == 'P') {p.first = i; p.second = j;}
 if (M[i][j] == 'F') F.push(make_pair(i, j));
 }
 }
 while (not F.empty()) {
 int i = (F.front ()). first ;
 int j = (F.front ()). second;
 F.pop();
 for (int k = 0; k < 8; ++k) M[i+xf[k]][j+yf[k]] = 'X';
 }
 if (cerca_bolet (M,p)) cout << "si" << endl;
 else cout << "no" << endl;
 }
}

```

## Solució de l'Examen d'Ordinador EDA

29/11/2012

## Proposta de solució al problema 1

(5 punts)

```
#include <iostream>
#include <sstream>
#include <vector>
#include <queue>

using namespace std;

typedef vector< queue<string> > VQS;

void read_queues(int n, VQS& v) {
 v = VQS(n);
 for (int k = 0; k < n; ++k) {
 string line;
 getline (cin, line);
 istringstream in(line);
 string name;
 while (in >> name) v[k].push(name);
 }
}

void process_exits (VQS& v) {
 cout << "SORTIDES" << endl;
 cout << "-----" << endl;

 string op;
 while (cin >> op) {
 if (op == "SURT") {
 int idx;
 cin >> idx;
 --idx;
 if (0 ≤ idx and idx < v.size () and not v[idx].empty()) {
 cout << v[idx].front () << endl;
 v[idx].pop ();
 }
 }
 else {
 string name;
 int idx;
 cin >> name >> idx;
 --idx;
 if (0 ≤ idx and idx < v.size ())
 v[idx].push(name);
 }
 }
 cout << endl;
}
```

```

}

void write_final_contents (VQS& v) {
 cout << "CONTINGUTS FINALS" << endl;
 cout << "-----" << endl;
 for (int k = 0; k < v.size (); ++k) {
 cout << "cua " << k+1 << ":";
 while (not v[k].empty()) {
 cout << ' ' << v[k].front ();
 v[k].pop ();
 }
 cout << endl;
 }
}

int main() {
 int n;
 cin >> n;
 string line ;
 getline (cin, line); // Read empty line.

 VQS v;
 read_queues (n, v);
 process_exits (v);
 write_final_contents (v);
}

```

## Proposta de solució al problema 2

(5 punts)

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

void compute_topological_ordering (const VVI& g, VI& indeg, VI& ord) {
 int n = g.size ();
 priority_queue <int, vector<int>, greater<int> > pq;
 for (int u = 0; u < n; ++u)
 if (indeg[u] == 0) pq.push(u);

 ord = VI(n);
 int cnt = 0;
 while (not pq.empty()) {
 int u = pq.top ();

```

```

 pq.pop();
 ord[cnt] = u;
 ++cnt;
 for (int k = 0; k < g[u].size (); ++k) {
 int v = g[u][k];
 --indeg[v];
 if (indeg[v] == 0) pq.push(v);
 }
}

}

void write(const VI& v) {
 cout << v[0];
 for (int k = 1; k < v.size (); ++k)
 cout << ' ' << v[k];
 cout << endl;
}

int main() {
 int n, m;
 while (cin >> n >> m) {
 VVI g(n);
 VI indeg(n, 0);
 for (int k = 0; k < m; ++k) {
 int u, v;
 cin >> u >> v;
 g[u].push_back(v);
 ++indeg[v];
 }
 VI ord;
 compute_topological_ordering (g, indeg, ord);
 write(ord);
 }
}

```

## Solució de l'Examen d'Ordinador EDA

22/5/2013

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> VI;

int n, m;
VI div;

bool ok(int y) {
 for (int i = 0; i < m; ++i)
 if (y % div[i] == 0)
 return false;
 return true;
}

void backtracking(int k, int x) {
 if (k == n) cout << x << endl;
 else {
 for (int v = 0; v ≤ 9; ++v) {
 int y = 10*x + v;
 if (ok(y)) backtracking(k+1, y);
 }
 }
}

int main() {
 while (cin >> n >> m) {
 div = VI(m);
 for (int i = 0; i < m; ++i)
 cin >> div[i];
 backtracking(0, 0);
 cout << "-----" << endl;
 }
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

```

```

typedef vector<char> VC; typedef vector< int> VI;
typedef vector< VC > VVC; typedef vector< VI > VVI;

typedef pair<int,int> P;

const int DF[] = {1, 1,-1,-1, 2, 2,-2,-2};
const int DC[] = {2,-2, 2,-2, 1,-1, 1,-1};

const int oo = 200 * 200;

bool ok(int i, int j, int n, int m, const VVC& t) {
 return i ≥ 0 and i < n and j ≥ 0 and j < m and t[i][j] ≠ 'X';
}

int distance (int f0, int c0, const VVC& t) {
 int n = t .size ();
 int m = t [0].size ();
 VVI d(n, VI(m, +oo));
 queue<P> q;
 q.push(P(f0, c0));
 d[f0][c0] = 0;
 while (not q.empty()) {
 P p = q.front ();
 q.pop ();
 int f = p.first ;
 int c = p.second;
 if (t[f][c] == 'p') return d[f][c];
 for (int k = 0; k < 8; ++k) {
 int i = f + DF[k];
 int j = c + DC[k];
 if (ok(i, j, n, m, t) and d[i][j] == +oo) {
 q.push(P(i, j));
 d[i][j] = d[f][c] + 1;
 }
 }
 }
 return +oo;
}

int main() {
 int n, m;
 while (cin >> n >> m) {
 VVC t(n, VC(m));

 for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j)
 cin >> t[i][j];
 }
}

```

```
int f0, c0;
cin >> f0 >> c0;
int dist = distance(f0-1, c0-1, t);
if (dist == +oo) cout << "no" << endl;
else cout << dist << endl;
}
}
```



## Solució de l'Examen d'Ordinador EDA

4/12/2013

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const int UNDEF = -1;
const int N_DIRS = 4;
const int DI[N_DIRS] = { 1, 0, -1, 0};
const int DJ[N_DIRS] = { 0, 1, 0, -1};

struct Pos {
 int i, j;
 Pos(int ii, int jj) : i(ii), j(jj) {}
};

bool ok(int n, int m, int i, int j) {
 return
 0 ≤ i and i < n and
 0 ≤ j and j < m;
}

int bfs(const vector < vector<char> >& map, int i0, int j0) {
 int n = map.size ();
 int m = map[0].size ();
 queue<Pos> q;
 q.push(Pos(i0, j0));
 vector< vector<int> > dist(n, vector<int>(m, UNDEF));
 dist[i0][j0] = 0;
 while (not q.empty()) {
 Pos p = q.front ();
 q.pop ();
 int i = p.i;
 int j = p.j;
 if (map[i][j] == 't') return dist[i][j];
 else {
 for(int k = 0; k < N_DIRS; ++k) {
 int ii = i + DI[k];
 int jj = j + DJ[k];
 if (ok(n, m, ii, jj) and map[ii][jj] ≠ 'X' and dist[ii][jj] == UNDEF) {
 q.push(Pos(ii, jj));
 dist[ii][jj] = 1 + dist[i][j];
 }
 }
 }
 }
}

```

```

 }
 return UNDEF;
}

int main(void) {

 int n, m;
 cin >> n >> m;

 vector < vector<char> > map(n, vector<char>(m));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j)
 cin >> map[i][j];

 int f, c;
 cin >> f >> c;

 int dist = bfs(map, f-1, c-1);
 if (dist > 0) cout << "distancia minima: " << dist << endl;
 else cout << "no es pot arribar a cap tresor" << endl;
}

```

### Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <map>
#include <assert.h>

using namespace std;

typedef map<string,int> MSI;
typedef vector<string> VS;
typedef vector<bool> VB;
typedef vector<int> VI;
typedef vector<VI> VVI;

int n_outputs, n_inputs;
VVI gdir, ginv;
MSI s2v;
VS v2s;

int string2vertex (const string& s) {
 auto i = s2v.find(s);
 if (i != s2v.end())
 return i->second;
 else {
 int v = v2s.size ();

```

```

 v2s.push_back(s);
 s2v.insert(make_pair(s, v));
 return v;
}
}

int main() {

 n_outputs = n_inputs = 0;

 string token;

 cin >> token;
 assert(token == "OUTPUT");
 while (cin >> token and token != "END") {
 ++n_outputs;
 string2vertex(token);
 }

 cin >> token;
 assert(token == "INPUT");
 while (cin >> token and token != "END") {
 ++n_inputs;
 string2vertex(token);
 }

 while (cin >> token and token != "END") {

 string s;
 cin >> s;
 int ov = string2vertex(s);
 if (ov+1 > gdir.size()) gdir.resize(ov+1);

 cin >> s;
 int iv1 = string2vertex(s);
 if (iv1 + 1 > ginv.size()) ginv.resize(iv1 + 1);

 if (token == "NOT") {
 gdir[ov].push_back(iv1);
 ginv[iv1].push_back(ov);
 }
 else {
 cin >> s;
 int iv2 = string2vertex(s);
 if (iv2 + 1 > ginv.size()) ginv.resize(iv2 + 1);
 if (token == "AND") {
 gdir[ov].push_back(min(iv1, iv2));
 gdir[ov].push_back(max(iv1, iv2));
 }
 }
 }
}

```

```

 else {
 gdir[ov].push_back(max(iv1,iv2));
 gdir[ov].push_back(min(iv1,iv2));
 }
 ginv[iv1].push_back(ov);
 ginv[iv2].push_back(ov);
}
}

int n = gdir.size ();
VI ddir(n, 0);
for (int v = 0; v < n; ++v)
 ddir[v] = gdir[v].size ();

VI bag;
for (int v = n_outputs; v < n_inputs + n_outputs; ++v)
 bag.push_back(v);

VI ord;
while (not bag.empty()) {
 int v = bag.back ();
 ord.push_back(v);
 bag.pop_back ();
 for (auto w : ginv[v]) {
 --ddir[w];
 if (ddir[w] == 0)
 bag.push_back(w);
 }
}

VB val(n);
while (cin >> token) {
 val[n_outputs] = (token == "T");
 for (int v = n_outputs+1; v < n_inputs + n_outputs; ++v) {
 cin >> token;
 val[v] = (token == "T");
 }

 for (auto v : ord) {
 if (gdir[v].size () == 1) {
 val[v] = not val[gdir[v][0]];
 }
 else if (gdir[v].size () == 2) {
 int iv1 = gdir[v][0];
 int iv2 = gdir[v][1];
 if (iv1 < iv2) val[v] = val[iv1] and val[iv2];
 else val[v] = val[iv1] or val[iv2];
 }
 }
}

```

```
 cout << (val[0] ? 'T' : 'F');
 for (int v = 1; v < n_outputs; ++v)
 cout << ' ' << (val[v] ? 'T' : 'F');
 cout << endl;
}
```

## Solució de l'Examen d'Ordinador EDA

19/5/2014

## Proposta de solució al problema 1

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<bool> VB;
typedef vector<int> VI;
typedef vector<VI> VVI;

void path(int xi, int xf, const VI& p) {
 if (xf == xi) cout << xi;
 else {
 path(xi, p[xf], p);
 cout << " " << xf;
 }
}

void bfs(const VVI& g, int xi, int xf) {
 int n = g.size ();
 VI p(n, -1);
 VB mkd(n, false);
 VI cur, pos;

 cur.push_back(xi);
 mkd[xi] = true;
 while (not cur.empty()) {
 for (int x : cur) {
 if (x == xf) {
 path(xi, xf, p);
 cout << endl;
 return;
 }
 for (int y : g[x])
 if (not mkd[y]) {
 pos.push_back(y);
 mkd[y] = true;
 p[y] = x;
 }
 }
 swap(pos, cur);
 }
}

int main() {
```

```

int n, m;
while (cin >> n >> m) {
 VVI g(n);
 for (int k = 0; k < m; ++k) {
 int x, y;
 cin >> x >> y;
 g[x].push_back(y);
 }
 for (int x = 0; x < n; ++x)
 sort(g[x].begin(), g[x].end());

 bfs(g, 0, n-1);
}

```

### Proposta de solució al problema 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<bool> VB;

void bt(int k, int n, string& w, VB& mkd) {
 if (k == n) cout << w << endl;
 else {
 for (int i = 0; i < n; ++i)
 if (not mkd[i] and (k == 0 or 'a' + i ≠ w[k-1] + 1)) {
 mkd[i] = true;
 w[k] = 'a' + i;
 bt(k+1, n, w, mkd);
 mkd[i] = false;
 }
 }
}

int main() {
 int n;
 cin >> n;
 string w(n, 'a');
 VB mkd(n, false);
 bt(0, n, w, mkd);
}

```

## Solució de l'Examen d'Ordinador EDA

22/12/2014

## Proposta de solució al problema 1

```
#include <iostream>
#include <vector>

using namespace std;

typedef vector<bool> VB;
typedef vector<int> VI;
typedef vector<VI> VVI;

const int UNDEF = -1;

bool cyclic (int x, const VVI& g, VB& mkd, VI& par) {
 if (mkd[x]) return true;
 mkd[x] = true;
 for (int y : g[x])
 if (par[x] != y) {
 par[y] = x;
 if (cyclic (y, g, mkd, par)) return true;
 }
 return false;
}

int nombre_arbres(const VVI& g) {
 int n = g.size ();
 VB mkd(n, false);
 VI par(n, UNDEF);
 int n_arb = 0;
 for (int x = 0; x < n; ++x) {
 if (not mkd[x]) {
 if (cyclic (x, g, mkd, par)) return UNDEF;
 else ++n_arb;
 }
 }
 return n_arb;
}

int main() {
 int n, m;
 while (cin >> n >> m) {
 VVI g(n);
 for (int k = 0; k < m; ++k) {
 int x, y;
 cin >> x >> y;
 g[x].push_back(y);
 g[y].push_back(x);
 }
 }
}
```



```

 }
 int n_arb = nombre_arbres(g);
 if (n_arb == UNDEF) cout << "no" << endl;
 else cout << n_arb << endl;
}
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> VI;

int bt(int k, const VI& m, int x, int sum_par, int max_und) {
 if (sum_par > x or sum_par + max_und < x) return 0;
 if (k == m.size()) return 1;
 int cnt = 0;
 for (int v = 0; v ≤ 2; ++v)
 cnt += bt(k+1, m, x, sum_par + v*m[k], max_und - 2*m[k]);
 return cnt;
}

int main() {
 int x, n;
 while (cin >> x >> n) {
 VI m(n);
 int s = 0;
 for (int k = 0; k < n; ++k) {
 cin >> m[k];
 s += m[k];
 }
 cout << bt(0, m, x, 0, 2*s) << endl;
 }
}

```

## Solució de l'Examen d'Ordinador EDA

25/05/2015

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> VC;
typedef vector<VC> VVC;

int bt(int i, int j, VVC& sol, int curr) {
 int n = sol.size();
 int m = sol[0].size();
 if (i == n) return curr;
 int next_i, next_j;
 if (j == m-1) {
 next_i = i+1;
 next_j = 0;
 }
 else {
 next_i = i;
 next_j = j+1;
 }

 sol[i][j] = 'L';
 int new_Jols = 0;
 if (i >= 2 and sol[i-1][j] == 'O' and sol[i-2][j] == 'L') ++new_Jols;
 if (j >= 2 and sol[i][j-1] == 'O' and sol[i][j-2] == 'L') ++new_Jols;
 if (i >= 2 and j >= 2 and sol[i-1][j-1] == 'O' and sol[i-2][j-2] == 'L') ++new_Jols;
 if (i >= 2 and j+2 < m and sol[i-1][j+1] == 'O' and sol[i-2][j+2] == 'L') ++new_Jols;
 int nl = bt(next_i, next_j, sol, curr + new_Jols);

 sol[i][j] = 'O';
 int no = bt(next_i, next_j, sol, curr);

 return max(nl, no);
}

int main() {
 int n, m;
 while (cin >> n >> m) {
 VVC sol(n, VC(m));
 cout << bt(0, 0, sol, 0) << endl;
 }
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<char> VC;
typedef vector<VC> VVC;
typedef vector<bool> VB;
typedef vector<VB> VVB;

bool ok(int i, int j, const VVC& t) {
 int n = t.size();
 int m = t[0].size();
 return i ≥ 0 and i < n and j ≥ 0 and j < m and t[i][j] ≠ 'X';
}

const int di[] = {0, 1, 0, -1};
const int dj[] = {1, 0, -1, 0};

bool possible(int i_ini, int j_ini, int i_fin, int j_fin, const VVC& t, VVB& mkd) {
 if (mkd[i_ini][j_ini]) return false;
 mkd[i_ini][j_ini] = true;
 if (i_ini == i_fin and j_ini == j_fin) return true;
 for (int k = 0; k < 4; ++k) {
 int i = i_ini + di[k];
 int j = j_ini + dj[k];
 if (ok(i, j, t) and possible(i, j, i_fin, j_fin, t, mkd)) return true;
 }
 return false;
}

int main() {
 int n, m;
 while (cin >> n >> m) {
 VVC t(n, VC(m));
 int i_ini, j_ini, i_fin, j_fin;
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j)
 cin >> t[i][j];

 for (int i = 0; i < n; ++i)
 for (int j = 0; j < m; ++j)
 if (t[i][j] == 'I') {
 i_ini = i;
 j_ini = j;
 }
 else if (t[i][j] == 'F') {
 i_fin = i;
 j_fin = j;
 }
 }
}

```

```

 }
 else if (t[i][j] == 'M') {
 t[i][j] = 'X';
 if (i-1 ≥ 0 and t[i-1][j] ≠ 'M') t[i-1][j] = 'X';
 if (i+1 < n and t[i+1][j] ≠ 'M') t[i+1][j] = 'X';
 if (j-1 ≥ 0 and t[i][j-1] ≠ 'M') t[i][j-1] = 'X';
 if (j+1 < m and t[i][j+1] ≠ 'M') t[i][j+1] = 'X';
 }

 VVB mkd(n, VB(m, false));
 if (possible (i_ini, j_ini, i_fin, j_fin, t, mkd)) cout << "SI" << endl;
 else cout << "NO" << endl;
}
}

```

## Solució de l'Examen d'Ordinador EDA

22/12/2015

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

const vector<pair<int, int>> DIRS = { {1, 0}, {-1, 0}, {0, 1}, {0, -1} };
const int UNDEF = -1;

int dist_2on_tresor_mes_llunya (int i0, int j0, vector<vector<char>>& mapa) {
 int max_dist = UNDEF;
 int max_dist2 = UNDEF;
 queue<pair<pair<int, int>, int>> q;
 q.push({{i0, j0}, 0});
 mapa[i0][j0] = 'X';
 while (not q.empty()) {
 int i = q.front().first.first;
 int j = q.front().first.second;
 int d = q.front().second;
 q.pop();
 for (auto dir : DIRS) {
 int ii = i + dir.first;
 int jj = j + dir.second;
 if (mapa[ii][jj] != 'X') {
 if (mapa[ii][jj] == 't') {
 max_dist2 = max_dist;
 max_dist = d+1;
 }
 q.push({{ii, jj}, d+1});
 mapa[ii][jj] = 'X';
 }
 }
 }
 return max_dist2;
}

int main() {
 int n, m;
 cin >> n >> m;
 vector<vector<char>> mapa(n+2, vector<char>(m+2, 'X'));
 for (int i = 1; i <= n; ++i)
 for (int j = 1; j <= m; ++j)
 cin >> mapa[i][j];

 int f, c;

```

```

cin >> f >> c;

int d2 = dist_2on_tresor_mes_llunya (f, c, mapa);
if (d2 == UNDEF) cout << "no es pot arribar a dos o mes tresors" << endl;
else cout << "segona distancia maxima: " << d2 << endl;
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>

using namespace std;

void escriu (int f, int c, const vector<int>& col) {
 for (int i = 0; i < f; ++i) {
 for (int j = 0; j < c; ++j)
 if (col[i] == j) cout << "R";
 else cout << ".";
 cout << endl;
 }
 cout << endl;
}

void torres (int f, int c, int i, vector<int>& col, vector<bool>& marked) {
 if (i == f) escriu (f, c, col);
 else
 for (int j = 0; j < c; ++j)
 if (not marked[j]) {
 col[i] = j;
 marked[j] = true;
 torres (f, c, i+1, col, marked);
 marked[j] = false;
 }
}

int main() {
 int f, c;
 cin >> f >> c;
 vector<int> col(f);
 vector<bool> marked(c, false);
 torres (f, c, 0, col, marked);
}

```

## Solució de l'Examen d'Ordinador EDA

19/05/2016

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

typedef pair<int,int> P;

int main(void) {
 int n, m;
 while (cin >> n >> m) {
 vector<vector<P>>> g(n);
 for (int k = 0; k < m; ++k) {
 int x, y, c;
 cin >> x >> y >> c;
 --x;
 --y;
 g[x].push_back(P(c, y));
 g[y].push_back(P(c, x));
 }
 vector<bool> mkd(n, false);
 mkd[0] = true;
 priority_queue<P, vector<P>, greater<P>> > pq;
 for (P x : g[0]) pq.push(x);
 int sz = 1;
 int sum = 0;
 while (sz < n) {
 int c = pq.top().first;
 int x = pq.top().second;
 pq.pop();
 if (not mkd[x]) {
 mkd[x] = true;
 for (P y : g[x]) pq.push(y);
 sum += c;
 ++sz;
 }
 }
 cout << sum << endl;
 }
}

```

## Proposta de solució al problema 2

```
#include <vector>

using namespace std;

// Pre: $l \leq r$, $x < v[r]$.
// Return the smallest i s.t. $l \leq i \leq r$ and $x < v[i]$.
int mes_a_la_dreta (double x, const vector<double>& v, int l, int r) {
 if (l == r) return l;
 int m = (l+r)/2;
 if (x < v[m]) return mes_a_la_dreta (x, v, l, m);
 else return mes_a_la_dreta (x, v, m+1, r);
}

int mes_a_la_dreta (double x, const vector<double>& v) {
 return mes_a_la_dreta (x, v, 0, v.size ());
}
```



## Solució de l'Examen d'Ordinador EDA

15/12/2016

Torn 1

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

vector<vector<int>>> su;
vector<vector<bool>>> r_mkd, c_mkd;
vector<vector<vector<bool>>>> s_mkd;

void write() {
 cout << endl;
 for (int i = 0; i < 9; ++i) {
 cout << su[i][0] + 1;
 for (int j = 1; j < 9; ++j)
 cout << ' ' << su[i][j] + 1;
 cout << endl;
 }
}

bool fill (int i, int j) {
 if (i == 9) return true;
 if (j == 9) return fill (i+1, 0);
 if (su[i][j] != -1) return fill (i, j+1);
 for (int v = 0; v < 9; ++v)
 if (not r_mkd[i][v] and not c_mkd[j][v] and not s_mkd[i/3][j/3][v]) {
 r_mkd[i][v] = c_mkd[j][v] = s_mkd[i/3][j/3][v] = true;
 su[i][j] = v;
 if (fill (i, j+1)) return true;
 r_mkd[i][v] = c_mkd[j][v] = s_mkd[i/3][j/3][v] = false;
 }
 su[i][j] = -1;
 return false;
}

int main() {
 su = vector<vector<int>>>(9, vector<int>(9));
 int n;
 cin >> n;
 cout << n << endl;
 while (n-- > 0) {
 r_mkd = c_mkd = vector<vector<bool>>>(9, vector<bool>(9, false));
 s_mkd = vector<vector<vector<bool>>>>(3, vector<vector<bool>>>(3, vector<bool>(9, false)));
 for (int i = 0; i < 9; ++i)
 for (int j = 0; j < 9; ++j) {

```

```

 char c;
 cin >> c;
 if (c == '.') su[i][j] = -1;
 else {
 int v = c - '1';
 r_mkd[i][v] = c_mkd[j][v] = s_mkd[i/3][j/3][v] = true;
 su[i][j] = v;
 }
}
if (fill(0, 0)) write();
}
}

```

### Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool search(const vector<int>& a, int l, int r, int x) {
 if (l+1 == r) return a[l] == x or a[r] == x;
 int m = (l+r)/2;
 if (a[m] ≥ a[l]) {
 if (a[l] ≤ x and x ≤ a[m]) {
 return binary_search(a.begin()+l, a.begin()+m+1, x);
 }
 else return search(a, m, r, x);
 }
 else {
 if (a[m] ≤ x and x ≤ a[r]) {
 return binary_search(a.begin()+m, a.begin()+r+1, x);
 }
 else return search(a, l, m, x);
 }
}

bool search(int x, const vector<int>& a) {
 return search(a, 0, a.size()-1, x);
}

```

**Torn 2****Proposta de solució al problema 1**

```

#include <iostream>
#include <vector>

using namespace std;

int m, n;
vector<int> l;

void gen(int i, int pos) {
 if (pos ≥ -m/2 and pos ≤ m/2) {
 if (i == n) cout << pos << endl;
 else {
 gen(i+1, pos+l[i]);
 gen(i+1, pos-l[i]);
 }
 }
}

int main() {
 cin >> m >> n;
 l = vector<int>(n);
 for (int& x : l) cin >> x;
 gen(0, 0);
}

```

**Proposta de solució al problema 2**

```

#include <iostream>

using namespace std;

typedef int Matrix [2][2];

void product(int m, const Matrix& a, const Matrix& b, Matrix& c) {
 for (int i = 0; i < 2; ++i)
 for (int j = 0; j < 2; ++j) {
 c[i][j] = 0;
 for (int k = 0; k < 2; ++k)
 c[i][j] = (c[i][j] + a[i][k] * b[k][j]) % m;
 }
}

void identity (Matrix& b) {
 b[0][0] = b[1][1] = 1;
 b[0][1] = b[1][0] = 0;
}

```

```

}

void power(int n, int m, const Matrix& a, Matrix& b) {

 if (n == 0)
 identity(b);
 else {
 if (n%2 == 0) {
 Matrix c;
 power(n/2, m, a, c);
 product(m, c, c, b);
 }
 else {
 Matrix c, d;
 power(n/2, m, a, c);
 product(m, c, c, d);
 product(m, a, d, b);
 }
 }
}

int main() {
 Matrix a;
 a[0][0] = 1; a[0][1] = 1;
 a[1][0] = 1; a[1][1] = 0;
 int n, m;
 while (cin >> n >> m) {
 Matrix b;
 power(n, m, a, b);
 cout << b[1][0] << endl;
 }
}

```

## Solució de l'Examen d'Ordinador EDA

25/05/2017

## Proposta de solució al problema 1

```
#include <iostream>
#include <vector>

using namespace std;

enum Letter {CON = 0, VOC};

void g(int k, int n, vector<string>& let, string& sol, vector<vector<bool>>& mkd) {
 if (k == 2*n) cout << sol << endl;
 else {
 int k2 = k%2;
 for (int i = 0; i < n; ++i)
 if (not mkd[k2][i]) {
 sol[k] = let[k2][i];
 mkd[k2][i] = true;
 g(k+1, n, let, sol, mkd);
 mkd[k2][i] = false;
 }
 }
}

int main() {
 int n;
 cin >> n;
 vector<vector<bool>> mkd(2, vector<bool>(n, false));
 vector<string> let(2);
 cin >> let[CON] >> let[VOC];
 string sol(2*n, ' '); // Need a char for filling.
 g(0, n, let, sol, mkd);
}
```

## Proposta de solució al problema 2

```
#include <iostream>
#include <vector>
#include <unordered_set>
#include <queue>

using namespace std;

const int UND = -1;

int main() {
 int n, t;
```

```

while (cin >> n >> t) {
 vector<unordered_set<int>> g(n);
 while (--t ≥ 0) {
 int s;
 cin >> s;
 vector<int> pub(s);
 for (int i = 0; i < s; ++i)
 cin >> pub[i];

 for (int i = 0; i < s; ++i)
 for (int j = i+1; j < s; ++j) {
 int u = pub[i];
 int v = pub[j];
 g[u].insert(v);
 g[v].insert(u);
 }
 }
 vector<int> dst(n, UND);
 queue<int> q;
 dst[0] = 0;
 q.push(0);
 while (not q.empty()) {
 int u = q.front();
 q.pop();
 for (int v : g[u])
 if (dst[v] == UND) {
 dst[v] = dst[u] + 1;
 q.push(v);
 }
 }
 for (int u = 0; u < n; ++u) {
 cout << u << " : ";
 if (dst[u] == UND) cout << "no";
 else cout << dst[u];
 cout << endl;
 }
 for (int k = 0; k < 10; ++k) cout << "-";
 cout << endl;
}
}

```

## Solució de l'Examen d'Ordinador EDA

25/05/2017

## Proposta de solució al problema 1

```
#include <iostream>
#include <vector>

using namespace std;

void write(int n, int p, const vector<string>& s, vector<int>& sol) {
 for (int j = 0; j < p; ++j) {
 cout << "subconjunt " << j+1 << ": {";
 string aux = "";
 for (int i = 0; i < n; ++i)
 if (sol[i] == j) {
 cout << aux << s[i];
 aux = ",";
 }
 cout << "}" << endl;
 }
 cout << endl;
}

void g(int k, int n, int p, const vector<string>& s, vector<int>& sol) {
 if (k == n) write(n, p, s, sol);
 else
 for (int i = 0; i < p; ++i) {
 sol[k] = i;
 g(k+1, n, p, s, sol);
 }
}

int main() {
 int n, p;
 cin >> n;
 vector<string> s(n);
 for (auto& x : s) cin >> x;
 cin >> p;
 vector<int> sol(n);
 g(0, n, p, s, sol);
}
```

## Proposta de solució al problema 2

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;
```

```

typedef pair<int,int> P;

int main() {
 int n, m;
 while (cin >> n >> m) {
 vector< vector<P> > g(n);
 int tot = 0;
 for (int k = 0; k < m; ++k) {
 int x, y, c;
 cin >> x >> y >> c;
 g[x].push_back({c, y});
 g[y].push_back({c, x});
 tot += c;
 }
 vector<bool> mkd(n, false);
 mkd[0] = true;
 priority_queue <P, vector<P>, greater<P> > pq;
 for (P p : g[0])
 pq.push(p);
 int sz = 1;
 int sum = 0;
 while (sz < n) {
 int c = pq.top (). first ;
 int x = pq.top (). second;
 pq.pop ();
 if (not mkd[x]) {
 mkd[x] = true;
 for (P p : g[x])
 pq.push(p);
 sum += c;
 ++sz;
 }
 }
 cout << tot - sum << endl;
 }
}

```



## Solució de l'Examen d'Ordinador EDA

28/05/2018

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

int n, m;
VVI g;
VI c;

const int UNDEF = -1;

// Returns true if no conflict is found.
bool propagate(int x, int col) {
 if (c[x] == UNDEF) {
 c[x] = col;
 for (int y: g[x])
 if (not propagate(y, 1-col)) return false;
 return true;
 }
 else return c[x] == col;
}

bool two_colorable(int x) {
 if (x == n) return true;
 if (c[x] == UNDEF) return propagate(x, 0) and two_colorable(x+1);
 else return two_colorable(x+1);
}

int main() {
 while (cin >> n >> m) {
 g = VVI(n);
 for (int k = 0; k < m; ++k) {
 int x, y;
 cin >> x >> y;
 g[x].push_back(y);
 g[y].push_back(x);
 }
 c = VI(n, UNDEF);
 if (two_colorable(0)) cout << "yes" << endl;
 else cout << "no" << endl;
 }
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Permutation {

 vector<int> v;

public:

 Permutation(int n) : v(n) { }
 Permutation(const Permutation& s) : v(s.v) { }

 Permutation& operator = (const Permutation& s) {
 v = s.v;
 return *this;
 }

 int& operator [](int i) { return v[i]; }
 int operator [](int i) const { return v[i]; }

 Permutation& operator *= (const Permutation& s) {
 for (int i = 0; i < v.size (); ++i)
 v[i] = s[v[i]];
 return *this;
 }

 void pow(int k) {
 if (k == 0)
 for (int i = 0; i < v.size (); ++i)
 v[i] = i;
 else {
 Permutation s = *this;
 s.pow(k/2);
 if (k % 2 == 0) {
 *this = s;
 *this *= s;
 }
 else {
 *this *= s;
 *this = s;
 }
 }
 }
};

```

```
int main() {
 int n;
 while (cin >> n) {
 Permutation p(n);
 for (int i = 0; i < n; ++i) {
 cin >> p[i];
 }
 int k;
 cin >> k;
 p.pow(k);

 for (int x = 0; x < n; ++x)
 cout << (x == 0 ? "" : " ") << p[x];
 cout << endl;
 }
}
```

## Solució de l'Examen d'Ordinador EDA

03/12/2018

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int top(const vector<int>& v, int l, int r) {
 if (l + 1 ≥ r) {
 if (v[l] < v[r]) return r;
 else return l;
 }
 int m = (l+r)/2;
 if (v[m-1] > v[m]) return top(v, l, m-1);
 if (v[m+1] > v[m]) return top(v, m+1, r);
 return m;
}

bool bin_search (bool inc, const vector<int>& v, int l, int r, int x) {
 if (l == r) return v[l] == x;
 int m = (l+r)/2;
 bool cond;
 if (inc) cond = (x ≤ v[m]);
 else cond = (x ≥ v[m]);
 if (cond) return bin_search (inc, v, l, m, x);
 else return bin_search (inc, v, m+1, r, x);
}

bool search(int x, const vector<int>& v) {
 int n = v.size ();
 int t = top(v, 0, n-1);
 return bin_search (true, v, 0, t, x) or bin_search (false, v, t, n-1, x);
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <queue>
#include <climits>

using namespace std;
using P = pair<int,int>;

const int oo = INT_MAX;

int cost (const vector<vector<P>>& g, const vector<int>& z, int a, int b) {

```

```

int n = g.size ();
vector<int> dist(n, +oo);
priority_queue <P, vector<P>, greater<P>> pq;
dist[a] = 0;
pq.push({0, a});
while (not pq.empty()) {
 auto t = pq.top ();
 pq.pop ();
 int d = t.first ;
 int u = t.second;
 if (d == dist[u]) {
 if (u == b) return dist[b];
 for (auto e : g[u]) {
 int v = e.first ;
 int w = e.second;
 int dv = dist[u] + w + (v == b ? 0 : z[v]);
 if (dist[v] > dv) {
 dist[v] = dv;
 pq.push({dv, v});
 }
 }
 }
}
return +oo;
}

int main() {
 int n, m;
 cin >> n >> m;
 vector<int> z(n);
 for (int& x : z) cin >> x;
 vector<vector<P>> g(n);
 while (m--) {
 int u, v, w;
 cin >> u >> v >> w;
 g[u].push_back({v, w});
 g[v].push_back({u, w});
 }
 int a, b;
 while (cin >> a >> b) {
 int c = cost(g, z, a, b);
 cout << "c(" << a << ", " << b << ") = ";
 if (c == +oo) cout << "+oo" << endl;
 else cout << c << endl;
 }
}

```

## Solució de l'Examen d'Ordinador EDA

20/05/2019

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

bool is_cyclic (const VVI& g) {
 int n = g.size ();
 VI indeg(n, 0);
 for (int u = 0; u < n; ++u)
 for (int v : g[u])
 ++indeg[v];

 vector<int> cands;
 for (int u = 0; u < n; ++u)
 if (indeg[u] == 0)
 cands.push_back(u);

 while (not cands.empty()) {
 int u = cands.back ();
 cands.pop_back ();
 --n;
 for (int v : g[u]) {
 --indeg[v];
 if (indeg[v] == 0)
 cands.push_back(v);
 }
 }
 return n > 0;
}

int main() {
 int n;
 while (cin >> n) {
 VVI g(n);
 int m;
 cin >> m;
 for (int k = 0; k < m; ++k) {
 int x, y;
 cin >> x >> y;
 g[x].push_back(y);
 }
 if (is_cyclic (g)) cout << "yes" << endl;
 }
}

```

```

 else cout << "no" << endl;
 }
}

```

### Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

void write(const vector<int>& x, const vector<bool>& sol) {
 cout << '{';
 bool first = true;
 for (int i = 0; i < int(x.size ()); ++i)
 if (sol[i]) {
 if (first) first = false;
 else cout << ',';
 cout << x[i];
 }
 cout << '}' << endl;
}

void bt(int s, int k, const vector<int>& x, int sp, int sn, vector<bool>& sol) {
 if (sp > s or sp + sn < s) return;
 if (k == int(x.size ())) write(x, sol);
 sol[k] = false; bt(s, k+1, x, sp, sn - x[k], sol);
 sol[k] = true; bt(s, k+1, x, sp + x[k], sn - x[k], sol);
}

int main() {
 int s, n;
 cin >> s >> n;
 vector<int> x(n);
 int sum = 0;
 for (int i = 0; i < n; ++i) {
 cin >> x[i];
 sum += x[i];
 }
 vector<bool> sol(n);
 bt(s, 0, x, 0, sum, sol);
}

```

## Solució de l'Examen d'Ordinador EDA

02/12/2019

## Proposta de solució al problema 1

```
#include <iostream>
#include <map>

using namespace std;

void addGame(const string& game, map<string,int>& m) {
 ++m[game];
}

bool canBuy(const string& game, map<string,int>& m, int totalGames){
 return (m[game] + 1 ≤ totalGames - m[game]);
}

int main(){
 int n;
 while (cin >> n) {

 map<string,int> m;
 int totalGames = 0;
 for (int i = 0; i < n; ++i) {
 string game; cin >> game;
 addGame(game,m);
 ++totalGames;
 }

 int g; cin >> g;
 for (int i = 0; i < g; ++i) {
 string game; cin >> game;
 if (canBuy(game,m,totalGames)){
 addGame(game,m);
 ++totalGames;
 }
 }

 for (auto& g:m) cout << g.first << " " << g.second << endl;
 cout << string(20, '-') << endl;
 }
}
```



## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <climits>
#include <queue>
using namespace std;

typedef vector<int> VI;
typedef vector<char> VC;
typedef vector<vector<char>> VVC;
typedef vector<vector<int>> VVI;
typedef pair<int,int> PII;

vector<PII> dirs = {{0,1},{0,-1},{1,0},{-1,0}};

bool pos_ok(const VVC& T, uint i, uint j){
 return i ≥ 0 and i < T.size () and j ≥ 0 and j < T[0].size () and T[i][j] ≠ '#';
}

PII cerca (const VVC& T) { // returns (distance,people)
 int n = T.size (), m = T[0].size ();
 VVI dist(n,VI(m,INT_MAX)), pers(n,VI(m,-1));
 queue<PII> Q;

 Q.push({0,0});
 dist [0][0] = 0;
 pers [0][0] = (T[0][0] == 'P');

 while (not Q.empty()) {
 int i = Q.front (). first , j = Q.front (). second;
 Q.pop();
 if (T[i][j] == 'T') return {dist [i][j], pers [i][j]};
 for (auto& d:dirs) {
 int ni = i + d.first , nj = j + d.second;
 int nd = dist [i][j] + 1;
 if (pos_ok(T,ni,nj)) {
 int np = pers [i][j] + int(T[ni][nj] == 'P');
 if (dist [ni][nj] == INT_MAX){
 Q.push({ni,nj});
 dist [ni][nj] = nd;
 pers [ni][nj] = np;
 }
 else if (dist [ni][nj] == nd and pers [ni][nj] < np)
 pers [ni][nj] = np;
 }
 }
 }
 return {-1,0};
}

```

```

int main(){
 int n, m;
 while (cin >> n >> m){
 VVC T(n,VC(m));
 bool found = false;
 for (int i = 0; i < n; ++i){
 for (int j = 0; j < m; ++j) {
 cin >> T[i][j];
 if (T[i][j] == 'T') found = true;
 }
 }
 if (not found) cout << "El telecos ha fugit." << endl;
 else {
 PII res = cerca(T);
 if (res.first == -1) cout << "El telecos esta amagat." << endl;
 else cout << res.first << " " << res.second << endl;
 }
 }
}

```

## Solució de l'Examen d'Ordinador EDA

10/06/2020

## Proposta de solució al problema 1

*Comprant accions, X39187:*

```
#include <iostream>
#include <vector>

using namespace std;

void rec(vector<char>& parcial, int idx, int monedes, int accions) {
 if (idx == int(parcial.size())) {
 for (auto& x : parcial) cout << x;
 cout << endl;
 }
 else {
 if (monedes > 0) {
 parcial[idx] = 'b';
 rec(parcial, idx+1, monedes-1, accions+1);
 }

 if (accions > 0) {
 parcial[idx] = 's';
 rec(parcial, idx+1, monedes+1, accions-1);
 }
 }
}

int main() {
 int n, c;
 cin >> n >> c;
 vector<char> parcial(n);
 rec(parcial, 0, c, 0);
}
```

*Paraules compensades, X46137:*

```
#include <iostream>
#include <vector>

using namespace std;

void rec(vector<char>& parcial, int idx, int a, int b) {
 if (idx == int(parcial.size())) {
 for (auto& x : parcial) cout << x;
 cout << endl;
 }
 else {
 if (abs((a+1)-b) ≤ 2) {
 parcial[idx] = 'a';
 rec(parcial, idx+1, a+1, b);
 }
 }
}
```

```

 }

 if (abs(a-(b+1)) ≤ 2) {
 parcial [idx] = 'b';
 rec (parcial ,idx+1,a,b+1);
 }
}
}

```

```

int main() {
 int n;
 cin >> n;
 vector<char> parcial(n);
 rec (parcial ,0,0,0);
}

```

*Pujades i baixades, X57029:*

```

#include <iostream>
#include <vector>

using namespace std;

void rec(vector<char>& parcial, int idx, int alcada_actual) {
 if (idx == int(parcial . size ())) {
 for (auto& x : parcial) cout << x;
 cout << endl;
 }
 else {
 if (alcada_actual ≠ 0) {
 parcial [idx] = 'd';
 rec (parcial ,idx+1,alcada_actual - 1);
 }

 parcial [idx] = 'h';
 rec (parcial ,idx+1,alcada_actual);

 parcial [idx] = 'u';
 rec (parcial ,idx+1,alcada_actual + 1);
 }
}

int main() {
 int n;
 cin >> n;
 vector<char> parcial(n);
 rec (parcial ,0,0);
}

```

**Proposta de solució al problema 2***Buscador de pel·lícules, X14417:*

```

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

vector<pair<int,int>> dirs = {{1,0},{-1,0},{0,1},{0,-1}};

bool ok(const vector<vector<string>>& M, int i, int j) {
 return i ≥ 0 and i < int(M.size()) and j ≥ 0 and j < int(M[0].size()) and M[i][j] ≠ "*";
}

int bfs(const vector<vector<string>>& M, pair<int,int>& orig, pair<int,int>& dest) {
 int n = M.size();
 int m = M[0].size();
 vector<vector<int>> dist(n,vector<int>(m,-1));
 queue<pair<int,int>> Q;

 Q.push(orig);
 dist[orig.first][orig.second] = 0;

 while (not Q.empty()) {
 pair<int,int> u = Q.front();
 Q.pop();
 if (u == dest) return dist[u.first][u.second];
 for (auto& d : dirs) {
 pair<int,int> v = u;
 v.first += d.first;
 v.second += d.second;
 if (ok(M,v.first,v.second) and dist[v.first][v.second] == -1) {
 Q.push(v);
 dist[v.first][v.second] = dist[u.first][u.second] + 1;
 }
 }
 }
 return -1;
}

pair<int,int> position(const vector<vector<string>>& M, const string& s){
 for (uint i = 0; i < M.size(); ++i){
 for (uint j = 0; j < M[0].size(); ++j){
 if (M[i][j] == s) return {i,j};
 }
 }
 return {-1,-1};
}

```

```

}

int main() {
 int n, m;
 while (cin >> n >> m) {
 vector<vector<string>>> M(n,vector<string>(m));
 for (int i = 0; i < n; ++i){
 for (int j = 0; j < m; ++j){
 cin >> M[i][j];
 }
 }
 int p;
 cin >> p;
 vector<string> paraules(p);
 for (int i = 0; i < p; ++i) cin >> paraules[i];

 vector<pair<int,int>>> positions(p+1);
 positions [0] = {0,0};
 for (int i = 0; i < p; ++i)
 positions [i+1] = position (M,paraules[i]);

 int total = 0;
 bool ok = true;
 for (uint i = 0; ok and i < positions . size () - 1; ++i) {
 int d = bfs (M,positions [i], positions [i+1]);
 if (d == -1) ok = false;
 else total += d;
 }
 if (ok) cout << (total + p) << endl;
 else cout << "impossible" << endl;
 }
}

```

Joc de cavall, X39759:

```

#include <iostream>
#include <vector>
#include <limits>
#include <queue>

using namespace std;

const int infinite = numeric_limits<int>::max();

vector<pair<int,int>>> dirs = {{1,2},{1,-2},{-1,2},{-1,-2},
 {2,1},{-2,1},{2,-1},{-2,-1}};

bool inside (int n, int m, const pair<int,int>& x){
 return x.first ≥ 0 and x.first < n and x.second ≥ 0 and x.second < m;
}

```

```

}

int bfs (int n, int m, const pair<int,int>& ini, const pair<int,int>& fi) {
 vector<vector<int>> dist(n,vector<int>(m,infinite));

 queue<pair<int,int>> Q;
 Q.push(ini);
 dist [ini . first][ini .second] = 0;

 while (not Q.empty()){
 pair<int,int> x = Q.front ();
 Q.pop();
 if (x == fi) return dist [x. first][x.second];
 for (auto& d : dirs) {
 pair<int,int> y = x;
 y. first += d. first ;
 y.second += d.second;
 if (inside (n,m,y) and dist[y. first][y.second] == infinite) {
 dist [y. first][y.second] = dist [x. first][x.second] + 1;
 Q.push(y);
 }
 }
 }
 return -1;
}

int main() {
 int n, m, W, L, p;
 while (cin >> n >> m >> W >> L >> p) {
 vector<pair<int,int>> objs(p+1);
 objs [0] = {0,0};
 for (int i = 1; i ≤ p; ++i) {
 int f, c;
 cin >> f >> c;
 objs [i] = {f,c};
 }

 int points = 0;
 int best_points = 0;
 bool stop = false;
 for (uint i = 0; not stop and i < objs . size () - 1; ++i) {
 int dist = bfs (n,m,objs [i], objs [i +1]);
 if (dist == -1) stop = true;
 else {
 points += W;
 points -= L*dist;
 if (points > best_points) best_points = points ;
 }
 }
 }
}

```

```
 cout << best_points << endl;
 }
}
```



## Solució de l'Examen d'Ordinador EDA

11/01/2021

Torn 1

## Proposta de solució al problema 1

```
#include <iostream>
#include <vector>
#include <stdlib.h>
```

```
using namespace std;
```

```
bool compatible(int left , int right , int d) {
 return abs(left -right) ≤ d;
}
```

```
void write_balanced_permutations (int n, int d, vector<int>& partial_sol, vector<bool>& used) {
 if (int(partial_sol .size ()) == n) {
 cout << "(";
 for (int i = 0; i < n; ++i) cout << (i == 0 ? "" : ",") << partial_sol [i];
 cout << ")" << endl;
 }
 else {
 for (int k = 1; k ≤ n; ++k) {
 if (not used[k]) {
 if (partial_sol .size () == 0 or compatible (partial_sol .back (),k,d)){
 partial_sol .push_back(k);
 used[k] = true;
 write_balanced_permutations (n,d, partial_sol ,used);
 used[k] = false;
 partial_sol .pop_back ();
 }
 }
 }
 }
}
```

```
void write_balanced_permutations (int n, int d) {
 vector<bool> used(n+1,false);
 vector<int> partial_sol ;
 write_balanced_permutations (n,d, partial_sol ,used);
}
```

```
int main(){
 int n, d;
 cin >> n >> d;
 write_balanced_permutations (n,d);
}
```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>

using namespace std;

void dfs(int u, const vector<vector<int>>& g, vector<int>& vis) {
 if (vis[u]) return;
 vis[u] = true;
 for (int v : g[u])
 dfs(v, g, vis);
}

int main() {
 int n, u, v, m;
 while (cin >> n >> u >> v >> m) {
 vector<vector<int>> g(n);
 vector<vector<int>> i(n); // Inverted graph (flipped edges)
 while (m-- > 0) {
 int x, y;
 cin >> x >> y;
 g[x].push_back(y);
 i[y].push_back(x);
 }
 vector<int> fwd(n, false);
 dfs(u, g, fwd);
 if (not fwd[v]) cout << 0 << endl;
 else {
 vector<int> bwd(n, false);
 dfs(v, i, bwd);
 int sum = 0;
 for (int x = 0; x < n; ++x) {
 if (fwd[x] and bwd[x]) cout << x << endl;
 sum += (fwd[x] and bwd[x]);
 }
 cout << sum - 2 << endl;
 }
 }
}

```

**Torn 2****Proposta de solució al problema 1**

```

#include <iostream>
#include <vector>

using namespace std;

bool compatible(int left , int mid, int right , int n) {
 return left + right ≤ 2*mid;
}

void write_no_well_permutations (int n, vector<int>& partial_sol, vector<bool>& used) {
 if (int(partial_sol .size ()) == n) {
 cout << "(";
 for (int i = 0; i < n; ++i) cout << (i == 0 ? "" : ",") << partial_sol [i];
 cout << ")" << endl;
 }
 else {
 for (int k = 1; k ≤ n; ++k) {
 if (not used[k]) {
 if (partial_sol .size () ≤ 1 or
 compatible(partial_sol [partial_sol .size ()-2], partial_sol .back (), k,n)){
 partial_sol .push_back(k);
 used[k] = true;
 write_no_well_permutations (n, partial_sol ,used);
 used[k] = false;
 partial_sol .pop_back ();
 }
 }
 }
 }
}

void write_no_well_permutations (int n) {
 vector<bool> used(n+1,false);
 vector<int> partial_sol ;
 write_no_well_permutations (n, partial_sol ,used);
}

int main(){
 int n;
 cin >> n;
 write_no_well_permutations (n);
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <queue>
#include <limits.h>

using namespace std;
using P = pair<int,int>;

const int oo = INT_MAX;

int dijkstra (const vector<vector<P>>& g, int x, int y) {
 int n = g.size ();
 vector<int> dist(n, +oo);
 priority_queue<P, vector<P>, greater<P>> q;
 dist[x] = 0;
 q.push({0, x});
 while (not q.empty()) {
 auto t = q.top ();
 q.pop ();
 int u = t.second;
 int d = t.first ;
 if (u == y) return dist[y];
 if (d == dist[u]) {
 for (auto p : g[u]) {
 int v = p.second;
 int l = p.first ;
 int d2 = max(dist[u], l);
 if (d2 < dist[v]) {
 dist[v] = d2;
 q.push({d2, v});
 }
 }
 }
 }
 return +oo;
}

int main() {
 int n, m;
 while (cin >> n >> m) {
 vector<vector<P>> g(n);
 while (m-->0) {
 int x, y, l;
 cin >> x >> y >> l;
 g[x].push_back({l, y});
 }
 cout << dijkstra (g, 0, 1) << endl; } }

```

## Solució de l'Examen d'Ordinador EDA

08/06/2021

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>

using namespace std;

void write_words(const vector<string>& words, vector<bool>& used, vector<int>& sol) {
 if (sol.size() == words.size()) {
 for (int idx : sol) cout << words[idx];
 cout << endl;
 }
 else {
 for (uint i = 0; i < words.size(); ++i) {
 if (not used[i] and (sol.size() == 0 or words[sol.back()].back() != words[i][0])) {
 used[i] = true;
 sol.push_back(i);
 write_words(words, used, sol);
 sol.pop_back();
 used[i] = false;
 }
 }
 }
}

void write_words(const vector<string>& words){
 int n = words.size();
 vector<bool> used(n, false);
 vector<int> sol; // solution will be a sequence of ints,
 // representing the indices in words
 write_words(words, used, sol);
}

int main(){
 int n;
 cin >> n;
 vector<string> words(n);
 for (string& s : words) cin >> s;

 write_words(words);
}

```

## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <queue>
#include <limits>

using namespace std;

typedef pair<int,int> Cell;
typedef vector<vector<int>>> Matrix;

const int inf = numeric_limits<int>::max();
const vector<pair<int,int>> dirs = {{0,-1},{0,1},{1,0},{-1,0}};

bool on_border (Cell& c, int n){
 return c.first == 0 or c.second == 0 or c.first == n-1 or c.second == n-1;
}

int cost (Matrix& M) {
 int n = M.size ();
 Cell center = {n/2,n/2};
 Matrix dist (n,vector<int>(n, inf));
 Matrix removed(n,vector<int>(n,0));

 priority_queue <pair<int,Cell>,vector<pair<int,Cell>>,greater<pair<int,Cell>>> Q;
 dist [center.first][center.second] = M[center.first][center.second];
 Q.push({dist[center.first][center.second], center});

 while (not Q.empty()) {
 Cell c = Q.top().second;
 Q.pop();
 if (not removed[c.first][c.second]) {
 removed[c.first][c.second] = 1;
 if (on_border(c,n)) return dist[c.first][c.second];
 for (auto& d : dirs) {
 Cell neigh = {c.first + d.first, c.second + d.second};
 if (dist[c.first][c.second] + M[neigh.first][neigh.second] < dist[neigh.first][neigh.second]) {
 dist[neigh.first][neigh.second] = dist[c.first][c.second] + M[neigh.first][neigh.second];
 Q.push({dist[neigh.first][neigh.second], neigh});
 }
 }
 }
 }
 return -1;
}

// Note that we can guarantee that neigh is always inside the
// grid. This is because we can only get out of the grid by visiting a
// neighbour of a border cell, but the algorithm stops as soon as we

```

// find such a cell

```
int main () {
 int n;
 while (cin >> n) {
 Matrix M(n,vector<int>(n));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < n; ++j)
 cin >> M[i][j];
 cout << cost(M) << endl;
 }
}
```

## Solució de l'Examen d'Ordinador EDA

07/01/2022

Torn 1

## Proposta de solució al problema 1

```

#include <iostream>
#include <vector>
#include <limits>
using namespace std;
int infinite = numeric_limits<int>::max();

// Returns size of connected component
// PRE: marked[u] = false
int size_con_component(const vector<vector<int>>& G, vector<bool>& marked, int u) {
 marked[u] = true;
 int total = 1;
 for (int v : G[u])
 if (not marked[v]) total += size_con_component(G, marked, v);
 return total;
}

// Returns (min,max)
pair<int,int> size_con_component(const vector<vector<int>>& G) {
 int n = G.size();
 pair<int,int> result = { infinite, 0 };
 vector<bool> marked(n, false);
 for (int u = 0; u < n; ++u) {
 if (not marked[u]) {
 int s = size_con_component(G, marked, u);
 result.first = min(result.first, s);
 result.second = max(result.second, s);
 }
 }
 return result;
}

int main() {
 int n, m;
 while (cin >> n >> m) {
 vector<vector<int>> G(n);
 for (int i = 0; i < m; ++i) {
 int x, y;
 cin >> x >> y;
 G[x].push_back(y);
 G[y].push_back(x);
 }
 pair<int,int> p = size_con_component(G);
 cout << p.first << " " << p.second << endl;
 }
}

```



## Proposta de solució al problema 2

```

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool is_vowel (char c) {
 return c == 'a' or c == 'e' or c == 'i' or c == 'o' or c == 'u';
}

// partial sol[0...k-1] already filled
void write_words (const vector<char>& letters, vector<char> partial_sol, int k) {
 if (k == int(partial_sol .size ())) {
 for (char c: partial_sol) cout << c;
 cout << endl;
 }
 else {
 for (char c : letters) {
 if (not is_vowel(c) or k == 0 or not is_vowel(partial_sol [k-1])) {
 partial_sol [k] = c;
 write_words(letters , partial_sol ,k+1);
 }
 }
 }
}

int main (){
 int n, m;
 while (cin >> n >> m) {
 vector<char> letters(m);
 for (int i = 0; i < m; ++i) cin >> letters [i];
 sort(letters .begin (), letters .end ());

 vector<char> partial_sol(n);
 write_words(letters , partial_sol , 0);
 cout << string(10,'-') << endl;
 }
}

```

**Torn 2****Proposta de solució al problema 1**

```

#include <iostream>
#include <vector>
#include <queue>
#include <limits>
using namespace std;
const int UNDEF = -1;

int farthest (const vector<vector<int>>& G) {
 int n = G.size ();
 vector<int> dist(n,UNDEF);
 queue<int> Q;
 Q.push(0);
 dist [0] = 0;
 while (not Q.empty()) {
 int u = Q.front ();
 Q.pop();
 for (int v : G[u]) {
 if (dist [v] == UNDEF) {
 dist [v] = dist [u] + 1;
 Q.push(v);
 }
 }
 }

 // This postprocessing could have been avoided by updating the max
 // distance every time we compute the distance for a new vertex. We
 // do it this way for simplicity and readability of the code.

 int result = 0;
 for (int v = 1; v < n; ++v)
 if (dist [v] != UNDEF and dist[v] > dist[result])
 result = v;
 return result ;
}

int main() {
 int n, m;
 while (cin >> n >> m){
 vector<vector<int>> G(n);
 for (int i = 0; i < m; ++i) {
 int x, y;
 cin >> x >> y;
 G[x].push_back(y);
 G[y].push_back(x);
 }
 cout << farthest (G) << endl;
 } }

```

**Proposta de solució al problema 2**

```
#include <iostream>
#include <vector>

using namespace std;

vector<char> letters = {'x', 'y', 'z'};

void write_words (int n, int c, vector<char>& partial_sol, int consec, int k) {
 if (k == n) {
 for (char aux : partial_sol) cout << aux;
 cout << endl;
 }
 else {
 for (int i = 0; i < 3; ++i) {
 bool repeated = (k > 0 and partial_sol [k-1] == letters [i]);
 int new_consec = (repeated ? consec + 1 : 1);
 if (new_consec ≤ c) {
 partial_sol [k] = letters [i];
 write_words(n,c, partial_sol ,new_consec,k+1);
 }
 }
 }
}

int main () {
 int n, c;
 while (cin >> n >> c) {
 vector<char> partial_sol(n);
 write_words(n,c, partial_sol ,0,0);
 cout << string(20,'-') << endl;
 }
}
```



---

## Solucions d'Exàmens Finals

## Solució de l'Examen Final EDA

18/01/2011

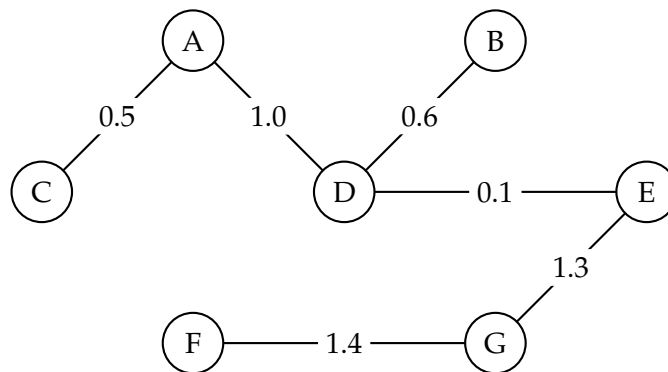
## Proposta de solució al problema 1

a) A, B, D, C, E, F, G, H, I, J, K, L, M.

b)

| $v[A]$ | $v[B]$   | $v[C]$   | $v[D]$   | $v[E]$   | $v[F]$   | $v[G]$   |
|--------|----------|----------|----------|----------|----------|----------|
| 0      | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0      | 1.5      | 0.5      | 1.0      | $\infty$ | $\infty$ | $\infty$ |
| 0      | 1.5      | 0.5      | 1.0      | $\infty$ | 6.8      | $\infty$ |
| 0      | 1.5      | 0.5      | 1.0      | 1.1      | 6.1      | 4.3      |
| 0      | 1.5      | 0.5      | 1.0      | 1.1      | 6.1      | 2.4      |
| 0      | 1.5      | 0.5      | 1.0      | 1.1      | 6.1      | 2.4      |
| 0      | 1.5      | 0.5      | 1.0      | 1.1      | 3.8      | 2.4      |

c)



## Proposta de solució al problema 2

Considereu que  $n$  és un enter positiu i que  $G$  és un graf de  $N$  nodes (vèrtexs) i  $E$  arestes (arcs) amb pesos.

- Primera afirmació. Cert. Les funcions de la forma  $f(n) = An^k$  (amb  $A$  i  $k$  constants) tenen un creixement polinòmic mentre que la funció  $2^n$  el té exponencial:  $\lim_{n \rightarrow \infty} \frac{An^k}{2^n} = 0$ .
- Segona afirmació. Cert. Tots els prefixos d'un camí mínim també són mínims. Altra-ment es podria trobar un camí alternatiu a  $C$  de cost menor (remplaçant el prefix no mínim pel mínim) contradient la hipòtesi de minimalitat de  $C$ .
- Tercera afirmació. Fals. Les  $N - 1$  arestes de menys pes de  $G$  poden formar un cicle i una d'elles quedar exclosa de l'arbre d'expansió mínim.
- Quarta afirmació. Fals. Considereu que  $G$  és tal que  $V = \{u, v, w\}$  i  $E = \{(u, v), (u, w), (v, w)\}$  cada arc amb pesos 1, 2.5 i 1 respectivament. El camí mínim de  $u$  a  $w$  és  $u, v, w$  però si s'afegeix 1 a cada arc aleshores seria  $u, w$ .

## Proposta de solució al problema 3

```

void escriu (Abc a) {
 if (a != NULL) {
 escriu (a -> fe);
 cout << a -> x << endl;
 }
}

```

```

 escriu (a -> fd);
 }
}

```

El cost és lineal.

#### Proposta de solució al problema 4

El primer problema és el problema de la PARTICIÓ, que sabem és **NP**-complet. Per tant, si  $P \neq NP$ , no pertany a **P**.

El segon problema es pot resoldre en temps  $O(n^3)$  amb l'algorisme evident. Per tant, pertany a **P** amb independència de la hipòtesi.

#### Proposta de solució al problema 5

En una taula de dispersió (hash)  $H$ , s'insereixen totes les arestes  $(u_i, v_i)$  de la llista  $E$  amb clau  $(u_i, v_i)$ .

A continuació, es fa un recorregut lineal de  $E$  per verificar si per cada aresta  $(u_i, v_i)$ , la clau  $(v_i, u_i)$  és a  $H$ .

El primer pas triga  $\Theta(m)$  en el cas mitjà, i com que en el cas mitjà cada operació de cerca a  $H$  té cost  $\Theta(1)$ , el segon pas triga  $\Theta(m)$  en mitjana.

#### Proposta de solució al problema 6

- Si  $d \geq 2$ ,  $T$  (l'arbre AVL més petit de profunditat  $d$ ) no pot ser buit i els seus fills  $T_E$  i  $T_D$  són arbres AVL i almenys un d'ells ha de tenir profunditat  $d - 1$ . Suposem que és  $T_E$ . Llavors  $T_D$  pot tenir profunditat  $d - 1$  o  $d - 2$ . Com que a més profunditat més nodes  $T_D$  ha de tenir profunditat  $d - 2$ , d'on es dedueix l'equació.
- L'algorisme demanat és pràcticament idèntic al que es fa servir per calcular els nombres de Fibonacci i es troba a la col·lecció d'algorismes de l'assignatura. El seu cost tant en temps com en espai és lineal. L'espai es pot reduir a constant si a l'aplicació no es repeteixen consultes amb valors inferiors de  $d$ .

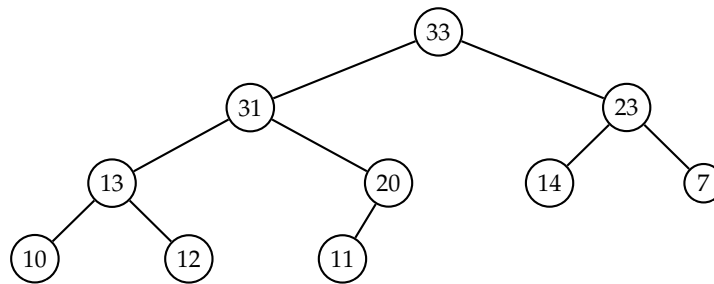
**Solució de l'Examen Final EDA****6/6/2011****Proposta de solució al problema 1**

- a) Els dos bucles tenen un cost  $\Theta(n^2)$  i es fan  $K$  crides recursives a desperdici( $n/2$ ). Si  $T(n)$  és el cost de desperdici amb entrada  $n$ , tenim que  $T(n) = KT(n/2) + \Theta(n^2)$ , amb  $n > 0$  i  $K > 0$ . Aplicant el teorema mestre,  $\alpha = \log_2(K)$  i per tant,

$$T(n) = \begin{cases} \Theta(n^2) & K < 4 \\ \Theta(n^2 \log(n)) & K = 4 \\ \Theta(n^\alpha) & K > 4 \end{cases}$$

- b) Sí és possible. Sigui  $T(n)$  el cost de l'algorisme que estem desenvolupant amb una entrada de mida  $n$  i  $g(n)$  el cost de dividir i ajuntar el problema de mida  $n$ . Aleshores  $T(n) = 3T(n/4) + g(n)$  amb  $g(n) = \Theta(n^k)$ . Aplicant el teorema mestre, si  $k > \log_4(3)$ , aleshores  $T(n) = \Theta(n^k)$ . Si  $k = 2$ ,  $T(n)$  seria de complexitat més gran que la desitjada, per tant la màxima  $k$  ha de ser  $k = 1$  i  $g(n) = \Theta(n)$ .

- c) 14, 20, 31



- d)

| $v[A]$ | $v[B]$   | $v[C]$   | $v[D]$   | $v[E]$   | $v[F]$   | $v[G]$   | $v[H]$   |
|--------|----------|----------|----------|----------|----------|----------|----------|
| 0      | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0      | 1        | $\infty$ | $\infty$ | 2        | 8        | $\infty$ | $\infty$ |
| 0      | 1        | 3        | $\infty$ | 2        | 7        | 7        | $\infty$ |
| 0      | 1        | 3        | $\infty$ | 2        | 7        | 7        | $\infty$ |
| 0      | 1        | 3        | 4        | 2        | 7        | 5        | $\infty$ |
| 0      | 1        | 3        | 4        | 2        | 7        | 5        | 8        |
| 0      | 1        | 3        | 4        | 2        | 6        | 5        | 7        |
| 0      | 1        | 3        | 4        | 2        | 6        | 5        | 7        |



e)

| $v[A]$ | $v[B]$   | $v[C]$   | $v[D]$   | $v[E]$   | $v[F]$   | $v[G]$   | $v[H]$   |
|--------|----------|----------|----------|----------|----------|----------|----------|
| 0      | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0      | 10       | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | 8        |
| 0      | 10       | $\infty$ | $\infty$ | $\infty$ | 12       | 9        | 8        |
| 0      | 5        | 10       | $\infty$ | $\infty$ | 8        | 9        | 8        |
| 0      | 5        | 6        | 11       | $\infty$ | 7        | 9        | 8        |
| 0      | 5        | 5        | 7        | 14       | 7        | 9        | 8        |
| 0      | 5        | 5        | 6        | 10       | 7        | 9        | 8        |
| 0      | 5        | 5        | 6        | 9        | 7        | 9        | 8        |

**Proposta de solució al problema 2**

- a) *misteri* (*Abc t* , *Elem x*) retorna el nombre de nodes de l'arbre binari de cerca *t* que són estrictament més petits que *x*.
- b) Si *n* és la mida de l'arbre, el cost en el cas pitjor és  $O(n)$ , ja que com a molt es visita cada node de l'arbre, i el treball a cada node costa temps constant. A més, això succeeix per exemple quan tots els elements de l'arbre són estrictament més petits que *x*. Per tant el cost és  $\Theta(n)$ .

c)

```

int misteri (Abc t , Elem x) {
 if (t == null) return 0;
 if (t->info < x) {
 if (t->fe == null) return 1 + misteri (t->fd, x);
 else return 1 + t->fe->m + misteri(t->fd, x);
 }
 return misteri (t->fe, x);
}

```

- d) El cost en el cas pitjor no millora perquè l'arbre pot no estar ben balancejat: per exemple, si l'arbre *t* és tal que tots els subarbres esquerres són nulls (és a dir, l'arbre de fet és una llista), i tots els elements de l'arbre són més petits que *x*, aleshores el cost continua sent lineal en la mida de l'arbre.
- e) El cost de la nova implementació de *misteri* en el cas pitjor és  $\Theta(h)$ , on *h* és l'alçada de l'arbre. Si l'arbre és un AVL, aleshores l'alçada és  $\Theta(\log(n))$ , on *n* és la mida de l'arbre. Per tant, en termes de *n*, el cost de *misteri* és  $\Theta(\log(n))$ .

**Proposta de solució al problema 3**

- a) `u == n`
- b) `usat[w] = false;`
- c) `G1[u][v] != G2[f[u]][f[v]]`
- d) `iso = backtracking(0);`

**Proposta de solució al problema 4**

- a) **Solució 1:** Com que CLIQUE és NP-complet, en particular és NP. I com que CLIQUE-3 és CLIQUE restringit a un subconjunt de les seves entrades, llavors CLIQUE-3 també és NP: els mateixos certificats i verificador serveixen.

**Solució 2:** Un candidat a certificat és un subconjunt de  $k$  vèrtexs del graf. Clarament té mida com a molt polinòmica (de fet, lineal) en l'entrada, i es pot verificar que realment és un certificat en temps polinòmic: només cal comprovar que tot parell de vèrtexs del subconjunt està connectat per una aresta.

- b) El raonament no permet concloure que CLIQUE-3 és NP-difícil. Per fer-ho, caldria reduir CLIQUE (o qualsevol altre problema NP-complet) en temps polinòmic a CLIQUE-3.

## Solució de l'Examen Final EDA

13/1/2012

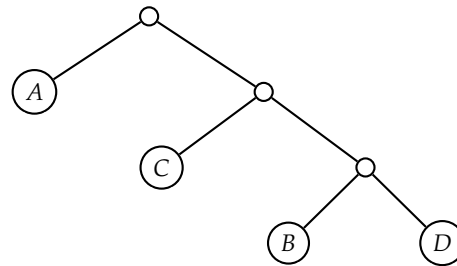
## Proposta de solució al problema 1

a)  $\Theta(n)$  en tots tres casos.

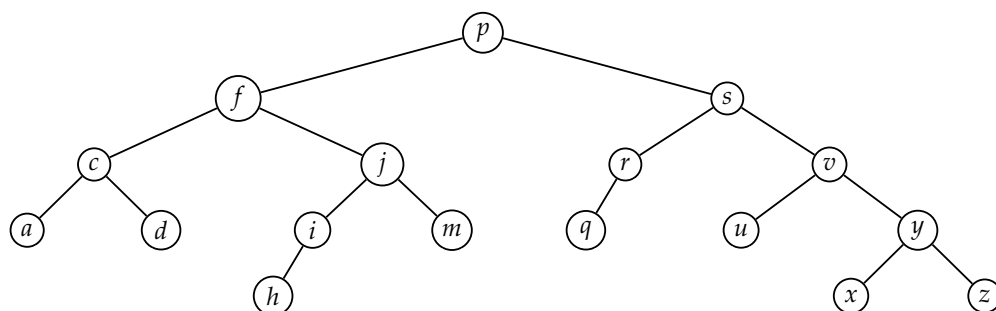
|   |   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 8 | 6 | 5 | 7 | 1 | 2 | 6 | 5 | 3 | 7 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

b)

|   |     |
|---|-----|
| A | 0   |
| B | 110 |
| C | 10  |
| D | 111 |



c)



d) Exponenciar la matriu  $M$  amb l'algorisme d'exponenciació ràpida, multiplicant matrius amb l'algorisme de Strassen i enters amb l'algorisme de Karatsuba i Ofman.

## Proposta de solució al problema 2

Compte! Funciona per pancakes però no per crêpes.

```

for (int i = n-1; i ≥ 0; --i) {
 int p = posicio del pancake mes gran entre 0 i i
 girar en el punt p
 girar en el punt i
}

```

L'algorisme és semblant a l'ordenació per selecció en el sentit que, a cada pas, es col·loca el següent element més gran a la posició que li pertoca. Com que es fan  $n$  iteracions i cada iteració necessita dos girs d'espàtula com a molt, s'ordenen  $n$  pancakes amb  $2n$  girs o menys.

### Proposta de solució al problema 3

```
*Node seguent(Abc a, Elem x) {
 if (not a) return a;
 if (x ≥ a -> info) return seguent(a -> fd, x);
 Node *p = seguent(a -> fe, x);
 if (p) return p;
 return a;
}
```

El cost és proporcional a l'alçada de l'arbre.

### Proposta de solució al problema 4

```
typedef pair<int, int> ArcP; // arc amb pes
typedef vector< vector<ArcP> > GrafP; // graf amb pesos

const int INFINIT = numeric_limits<int>::max();

void calcula (const GrafP& G, int s, vector<int>& d, vector<int>& p) {
 int n = G.size ();
 d = vector<int>(n, INFINIT);
 d[s] = 0;
 p = vector<int>(n, 0);
 p[s] = 1;
 vector<bool> S(n, false);
 priority_queue <ArcP, vector<ArcP>, greater<ArcP> > Q;
 Q.push(ArcP(0, s));
 while (not Q.empty()) {
 int u = Q.top ().second; Q.pop();
 if (not S[u]) {
 S[u] = true;
 for (int i = 0; i < int(G[u].size ()); ++i) {
 int v = G[u][i].first ;
 int c = G[u][i].second;
 if (d[v] > d[u] + c) {
 d[v] = d[u] + c;
 p[v] = p[u];
 Q.push(ArcP(d[v], v));
 }
 }
 else if (d[v] == d[u] + c) p[v] += p[u];
 }
 }
}

int main(void) {
```

```

int n, m;
while (cin >> n >> m) {
 GrafP G(n);
 for (int k = 0; k < m; ++k) {
 int u, v, c;
 cin >> u >> v >> c;
 G[u].push_back(ArcP(v,c));
 }
 int x, y;
 cin >> x >> y;
 vector<int> d,p;
 calcula (G, x, d, p);
 if (d[y] == INFINIT)
 cout << "No hi ha cami de " << x << " a " << y << endl;
 else
 cout << "Cost " << d[y] << ", " << p[y] << " manera(es)" << endl;
} }

```

### Proposta de solució al problema 5

Donada una instància  $(G = (V, E), k)$  del problema RECOBRIMENT volem transformar-la en una instància  $(k', P, \mathcal{C})$  del problema de CLAUS.

Fem  $k' = k$ ,  $P = E$  i  $C_i = \{\{v_i, v_j\} \in E \text{ per algun } v_j \in V\}$  on  $V = v_1 \dots v_n$  i  $\mathcal{C} = \{C_1, \dots, C_n\}$  (recordem que cal que  $C_i \subseteq P$ ).

Aleshores, si la instància de RECOBRIMENT és positiva, la instància corresponent de CLAUS també ho és. Efectivament, si existeix un  $A \subseteq V$  de  $k$  vèrtexs tal que  $\forall (u, v) \in E, u \in A \vee v \in A$ , podem construir  $\mathcal{C}$  amb els  $C_i$  que corresponen als  $v_i \in A$ . Així doncs,  $\forall p \in P$  cal veure que  $\exists C_j \in \mathcal{C}, p \in C_j$ , però  $p \equiv (u, v)$  i  $u \in A \vee v \in A$  per ser  $A$  un recobriment. Aleshores és fàcil veure que l'aresta  $p$  pertany a  $C_u$  i també a  $C_v$ .

A l'inrevés, donat  $\mathcal{C}$  triem els vèrtexs que es corresponen a les claus per formar  $A$ . Com tot pany (aresta) és obert al menys per una clau  $C_s$ , és clar que, per definició, o bé un extrem de  $p$  pertany a  $C_s$  o bé hi pertany l'altre extrem. Sigui com sigui, això és el mateix que dir que un dels dos extrems de  $p$  és a  $A$ , per construcció d' $A$ . Per tant, com  $p$  és en realitat qualsevol aresta,  $A$  és un recobriment.

Així acabem la demostració  $\text{RECOBRIMENT} \leq_p \text{CLAUS}$ , per tant CLAUS és NP-difícil.

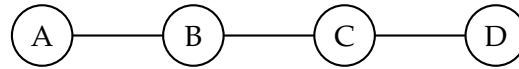
Per veure que CLAUS és a NP (i per tant concloure que CLAUS és NP-complet) caldria un testimoni per poder veure, en temps polinòmic, si una instància del problema és o no positiva. El testimoni pot ser el mateix conjunt  $\mathcal{C}$ . Per veure si la instància és o no positiva caldria comprovar que tot pany és obert per alguna clau  $C_i$ . Ho podem fer mirant que la unió de tots els  $C_i$  és  $P$ . Això òbviament es pot fer en temps polinòmic.

## Solució de l'Examen Final EDA

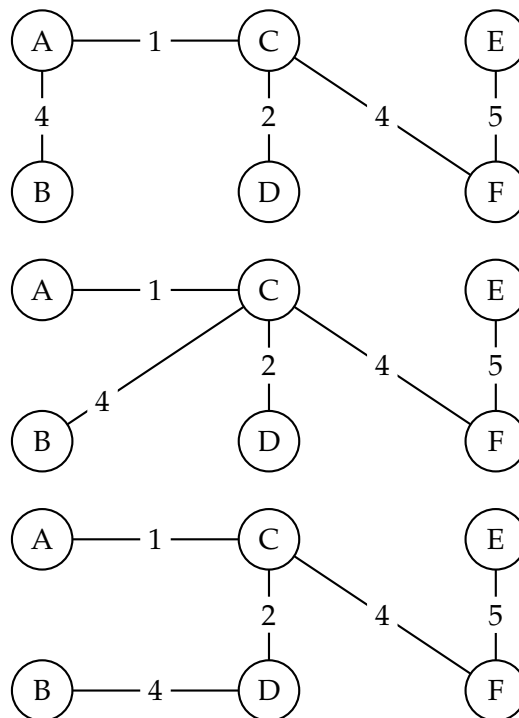
15/6/2012

## Proposta de solució al problema 1

a) El vèrtex d'inici és l'A.



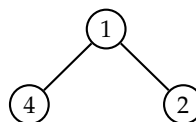
b) A continuació es mostren diversos arbres d'expansió mínims del graf de l'enunciat (i per tant, no hi ha unicitat):

c) Res. Com que el cost dels intercanvis és  $\Theta(1)$ , fent  $n = d - e + 1$  el cost del codi segueix la recurrència

$$T(n) = 3T(n/2) + \Theta(1),$$

i del Teorema Mestre de Recurrències Divisores obtenim que  $T(n) = \Theta(n^{\log_2 3})$ .

d) • FALS. Considerem el següent heap:



La seva representació en taula és:

|   |   |   |
|---|---|---|
| 1 | 4 | 2 |
|---|---|---|

L'element màxim (4) ocupa la 2a posició final. Però en aquest cas  $n = 3$  i  $\lfloor n/2 \rfloor = 1$ .

- FALS. En el cas pitjor quicksort és quadràtic, mentre que per exemple mergesort sempre és  $\Theta(n \log n)$ .
- CERT. Representem el conjunt de vèrtexs amb  $V$  i el grau d'un vèrtex  $v \in V$  amb  $g(v)$ . Llavors tenim que

$$2|E| = \sum_{v \in V} g(v) = \sum_{v \in V, g(v) \text{ parell}} g(v) + \sum_{v \in V, g(v) \text{ senar}} g(v),$$

d'on  $\sum_{v \in V, g(v) \text{ senar}} g(v) = 2|E| - \sum_{v \in V, g(v) \text{ parell}} g(v)$  ha de ser parell. Per tant, hi ha un nombre parell de sumands a  $\sum_{v \in V, g(v) \text{ senar}} g(v)$ , és a dir, hi ha un nombre parell de vèrtexos de grau senar.

- CERT. Per definició de problema decisional.

### Proposta de solució al problema 2

a)

```
void resize () {
 vector<vector<KeyInfo>>> aux(n);
 for (int i = 0; i < M; ++i) {
 for (int j = 0; j < v[i].size (); ++j) {
 Key& k = v[i][j];
 int p = hash_function(k) % n;
 aux[p].push_back(k);
 }
 }
 v.swap(aux);
 M = n;
}
```

- b) El cos del bucle intern i l'increment de  $j$  tenen cost  $\Theta(1)$  i s'executen  $\Theta(n)$  vegades en total. Per tant, contribueixen al cost total en  $\Theta(n)$ . Per altra banda, l'increment de  $i$ , la inicialització de  $j$  i l'avaluació de la condició del bucle més extern tenen cost  $\Theta(1)$ , i s'executen  $\Theta(M)$  vegades. Com que  $M = n/\alpha$  i  $\alpha > 2$ , tenim  $M = O(n)$  i que la contribució al cost total és  $O(n)$ . Pel que fa a l'avaluació de la condició del bucle més intern, contribueix al cost total en  $\Theta(n + M) = \Theta(n)$ . Finalment, la creació d'  $aux$  té cost  $\Theta(n)$ , i la resta del codi contribueix en cost constant. Així doncs, el cost és  $\Theta(n)$ .
- c) Cada cop que el nombre d'elements guardats a la taula és de la forma  $2^l - 1$ , amb  $l \geq 2$ , es fa una crida a *resize*. Per tant, si  $n$  està entre  $2^{k+1} - 1$  i  $2^{k+2} - 2$  per a un cert  $k$ , llavors es fan  $k$  crides a *resize*, la contribució al cost de les quals en el cas pitjor és, per l'apartat anterior,  $\sum_{l=2}^{k+1} \Theta(2^l - 1) = \sum_{l=2}^{k+1} \Theta(2^l) = \Theta(2^k) = \Theta(n)$ . Per altra banda, la resta del codi de *insert* costa  $\Theta(1)$  a cada crida. Per tant, després de  $n$  crides la contribució al cost és  $\Theta(n)$ . Finalment, el cost total és doncs  $\Theta(n)$ .

### Proposta de solució al problema 3

- a) **void** rotacio (Node\*& a, int x) {  
     **if** (x < a->x) **return** rotacio(a->fe, x);  
     **if** (x > a->x) **return** rotacio(a->fd, x);  
     Node\* b = a;  
     a = a->fe;

```

 b->fe = a->fd;
 a->fd = b;
}

```

b) El cost de *rotacio* () ve donat pel de trobar l'element a l'arbre, més el de fer la rotació. Aquest últim és constant i el primer és proporcional a l'alçada de l'arbre. En el cas pitjor, un ABC té alçada lineal i, per tant, el cost en el cas pitjor és  $\Theta(n)$ .

c) Per un raonament semblant a l'anterior el cost de *rotacio* () en un arbre AVL és  $\Theta(\log n)$ .

#### Proposta de solució al problema 4

```

class Particio {
 int n, min_dif, dif, total;
 vector<int> v;

 void recursiu (int i) {
 int va = valor_absolut (dif);
 if (va - total >= min_dif) return;
 if (i == n) min_dif = va;
 else {
 dif += v[i]; total -= v[i];
 recursiu (i+1);
 dif -= 2*v[i];
 recursiu (i+1);
 dif += v[i]; total += v[i];
 }
 }

public:
 Particio (int n, int total, vector<int>& v) {
 this->n = n;
 this->total = total;
 this->v = v;
 min_dif = total;
 dif = 0;
 recursiu (0);
 cout << min_dif << endl;
 }
};

int main() {
 int n;
 cin >> n;
 int total = 0;
 vector<int> v(n);
 for (int i = 0; i < n; ++i) {cin >> v[i]; total += v[i];}
 Particio p(n, total, v);
}

```



**Proposta de solució al problema 5**

- a) Com que  $X$  es pot reduir polinòmicament a  $A$  i  $A$  pertany a NP (per ser NP-complet), aleshores  $X$  pertany a NP. No obstant, no es pot afirmar que  $X$  sigui NP-complet.
- b) Per ser  $A$  NP-complet, tenim que  $A$  és NP-difícil. Per tant, tot problema de la classe NP es pot reduir polinòmicament a  $A$ . Com que per hipòtesi  $A$  es pot reduir polinòmicament a  $X$ , això implica que  $X$  és NP-difícil. No obstant, això no implica que  $X$  pertanyi a NP, i per tant tampoc implica que sigui NP-complet.
- c) Com que  $B$  pertany a NP, la resposta és la mateixa que a l'apartat a).
- d) A diferència de l'apartat b), en aquest cas no podem ni tan sols afirmar que  $X$  sigui NP-difícil. Tampoc podem afirmar que  $X$  pertanyi a NP, ni que sigui NP-complet.

## Solució de l'Examen Final EDA

10/01/2013

## Proposta de solució al problema 1

- A Fer una passada per a cada nombre fins tornar-lo a trobar. Temps  $O(n^2)$  en cas pitjor i espai addicional  $O(1)$ .
- B Ordenar els nombres i fer una passada fins trobar l'únic. Temps  $O(n \log n)$  en cas pitjor i espai addicional  $O(1)$  utilitzant Heapsort.
- C Utilitzar un conjunt d'enters. Es recorre cada element de la seqüència, esborrant-lo del conjunt si ja hi era, afegint-lo al conjunt si no hi era. Al final, només pot haver-hi un sol element al conjunt, que és la solució. Temps  $O(n \log n)$  i espai addicional  $O(n)$  amb arbres equilibrats. Temps  $O(n)$  en mitjana i espai addicional  $O(n)$  amb hashing.
- D Calcular la xor de tots els enters. El resultat és el nombre únic, perquè els bits dels repetits s'han cancel·lat. Temps  $O(n)$  i espai addicional  $O(1)$ .

## Proposta de solució al problema 2

- La solució a) té cost en el cas pitjor  $\Theta(n \log n)$ , mentre que la solució b) té cost  $\Theta(mn)$ . Així doncs, si  $m$  és  $O(\log n)$ , llavors la solució b) és almenys tan bona com la a); i si  $m$  és  $\Omega(\log n)$ , llavors la solució a) és almenys tan bona com la b).
- Seleccionar l'element  $m$ -èsim (amb cost lineal) i després fer una partició (à la quicksort) amb aquest element de pivot (també amb cost lineal). Així, a les primeres  $m$  posicions tindríem (no ordenats) els elements que busquem. El cost total en temps d'aquest algorisme és per tant  $\Theta(n)$ .

## Proposta de solució al problema 3

1. Un dígraf implementat amb la classe *Dígraf1* té cost en espai  $\Theta(n^2)$ , i un d'implementat amb la classe *Dígraf2* té cost en espai  $\Theta(n + m)$ . En qualsevol de les dues implementacions, qualsevol dígraf és cas pitjor.

2. (a) Per a la classe *Dígraf1*:

```
bool hi_ha_arc (int u, int v) { return t[u][v];}
```

Té cost en el cas pitjor  $\Theta(1)$ . Aquest cas pitjor es dona sempre.

- (b) Per a la classe *Dígraf2*:

```
bool hi_ha_arc (int u, int v) {
 for (int i = 0; i < t[u].size (); ++i)
 if (t[u][i] == v) return true;
 return false;
}
```

Té cost en el cas pitjor  $\Theta(\min(m, n))$ . Per exemple, aquest cas pitjor es dona:

- si  $m < n$ : quan totes les arestes surten de  $u$  i cap arriba a  $v$ .
- si  $m \geq n$ : quan  $u$  té com a successors tota la resta de vèrtexs menys  $v$ .

3. class *Dígraf3* {

```
vector< set<int> > t;
```

**public:**

```

 Digraf3(int n) { t = vector< set<int> >(n); }

 void afegeix_arc (int u, int v) { t[u].insert (v); }

 bool hi_ha_arc (int u, int v) {
 return t[u].find (v) != t[u].end ();
 }
};

```

#### Proposta de solució al problema 4

```

#include <vector>
#include <iostream>

using namespace std;

typedef vector<vector<int>>> Graph;

class ThreeColoring {

 Graph G;
 int n;
 vector<int> col;
 bool done;

 bool legal (int u, int c) {
 for (int v : G[u]) { // què bonic!
 if (v < u and col[v] == c) {
 return false;
 }
 }
 return true;
 }

 void rec(int u) {
 if (u == n) done = true
 else {
 for (int c = 0; c < 3 and not done; ++c) {
 if (legal (u, c)) {
 col[u] = c;
 rec(u+1);
 }
 }
 }
 }
};

```

**public:**

```

ThreeColoring(Graph G) : G(G), n(G.size ()), col(n), done(false) { rec (0); }

```

```

bool colorable () const { return done; }

int color (int u) { return col[u]; }

};

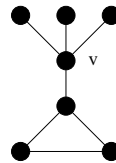
int main() {
 Graph G = { {1,2}, {0,3}, {0,3}, {1,2} };

 ThreeColoring col(G);
 cout << col.colorable () << endl;
}

```

### Proposta de solució al problema 5

(a) *Incorrecta*. Considerem el graf  $G$  següent:



El parell  $(G, 3)$  pertany a **CLICA** però, en canvi, la transformació que en fa  $f$  és  $(G, 3, v)$ , que no pertany a **CLICA-CENTRADA** perquè  $v$  no és part del triangle.

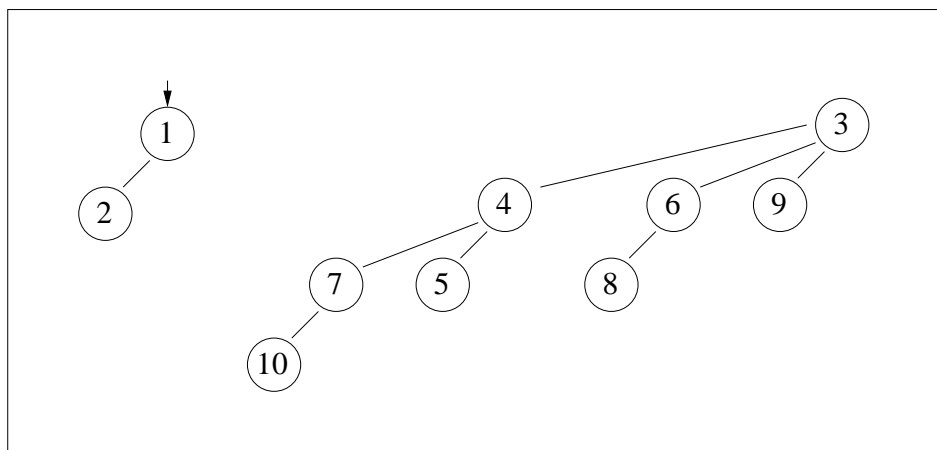
- (b) *Incorrecta*. En aquest cas,  $(G', k, w_1)$  sempre pertany a **CLICA-CENTRADA** i, per tant, la reducció no pot ser vàlida. Qualsevol parell  $(G, k)$  tal que  $G$  no tingui una  $k$ -clica serveix de contraexemple.
- (c) *Correcta*. És evident que si  $G$  té una  $k$ -clica amb conjunt de vèrtexs  $U$ , llavors  $G''$  té una  $(k + 1)$ -clica amb conjunt  $U \cup \{u\}$ . D'altra banda, si  $G''$  té una  $(k + 1)$ -clica que conté  $u$ , la còpia de  $G$  ha de tenir una  $k$ -clica.

**Solució de l'Examen Final EDA****5/6/2013****Proposta de solució al problema 1**

- a) Dominen els polinomis de grau 3, per tant  $\Theta(n^3)$ .
- b) Tenim  $\Theta(n^3) = \Theta((n^2)^{3/2}) = \Theta(N^{3/2})$  perquè  $N = \Theta(n^2)$ . Per tant la resposta és  $\Theta(N^{3/2})$ .
- c) Quan  $n$  es multiplica per 2, el temps es multiplica per  $2^3 = 8$ . Per tant 8 vegades 15 segons, és a dir 2 minuts.
- d) El factor  $k$  que ens demanen ha de ser tal que  $k^3 = 1000$ . És a dir,  $k = 10$ .

**Proposta de solució al problema 2**

El heap binomial resultant és el següent:



Per obtenir-lo, hem mantingut el  $B_1$  del primer heap i hem fusionat els dos  $B_3$  de cada heap. Aquest segon pas l'hem fet penjant el  $B_3$  que té l'arrel més gran de l'arrel de l'altre  $B_3$  per tal de preservar la propietat de min-heap.

**Proposta de solució al problema 3**

Sense pèrdua de generalitat, suposem que  $V = \{1, \dots, n\}$ .

**Apartat a.** Demostració per inducció sobre  $k$ .

*Cas base.* Per  $k = 1$  tenim que  $A^1 = A$  és la matriu d'adjacències de  $G$  i se satisfà que  $A[i, j] = 1$  si i només si hi ha camí de longitud  $\leq 1$  en  $G$ .

*Hipòtesi d'inducció.*  $A^i$  és la matriu d'adjacències de  $G^i$ , per a tot  $i$ ,  $1 \leq i \leq k$  essent  $k \geq 1$ .

Considerem ara la potència (booleana)  $(k+1)$ -èsima  $A^{k+1} = A^k A$ . Per la definició del producte (booleà) de matrius,  $A^{k+1}[i, j] = \bigvee_{\ell=1}^n (A^k[i, \ell] \wedge A[\ell, j])$ . Aleshores,  $A^{k+1}[i, j] = 1$  si i només si existeix  $\ell$ ,  $1 \leq \ell \leq n$  tal que  $A^k[i, \ell] = 1$  i  $A[\ell, j] = 1$ . Aplicant la hipòtesi d'inducció,  $A^k[i, \ell] = 1$  si i només si hi ha algun camí de  $i$  a  $\ell$  de longitud  $\leq k$  en  $G$ . Per tant,  $A^{k+1}[i, j] = 1$  si i només si per algun  $\ell$ ,  $1 \leq \ell \leq n$  hi ha algun camí de longitud  $\leq k$  en  $G$  des de  $i$  a  $\ell$  i hi ha algun camí de longitud  $\leq 1$  en  $G$  des de  $\ell$  a  $j$ . Aleshores,  $A^{k+1}[i, j] = 1$  si i només si hi ha algun camí de longitud  $\leq k+1$  en  $G$  des de  $i$  a  $j$  o el que és el mateix, si i només si  $(i, j) \in E^{k+1}$ .

**Apartat b.**

```

typedef vector<int> VB;
typedef vector<VB> VVB;

VVB Potencia_2(VVB& A){
 int n = A.size ();
 VVB B(n, VB(n, 0));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < n; ++j)
 for (int l = 0; l < n; ++l){
 B[i][j] = B[i][j] or (A[i][l] and A[l][j]);
 }
 return B;
}

VVB Potencia_4(VVB& A){
 A = Potencia_2(A); /* A matriu d'adjacencies de G^2 */
 A = Potencia_2(A); /* A matriu d'adjacencies de G^4 */
 return A;
}

```

Sigui  $A$  una matriu de 0s i 1s qualsevol de dimensió  $n \times n$  on  $n \geq 1$ . La funció `Potencia_2` amb entrada  $A$ , computa el quadrat booleà d' $A$ . El temps de computació és  $\Theta(n^3)$ . Aleshores, la funció `Potencia_4` amb entrada  $A$ , computa  $A^4$  (4a potència booleana) i el temps de computació de `Potencia_4` és també  $\Theta(n^3)$ .

**Apartat c.**

Si fem el producte de matrius d'enters habitual, sense canviar el producte d'enters per un AND ni la suma d'enters per un OR, llavors el resultat de  $A^k[i, j]$  serà 0 si i només si no hi ha cap camí de longitud menor o igual que  $k$  de  $i$  a  $j$ . Vist així, podem fer servir l'algorisme de Strassen per multiplicar dues matrius d'enters dues vegades, primer per obtenir  $A^2$  i després per obtenir  $A^4$ . El temps serà  $\Theta(n^{\log_2 7})$  i d'aquí en podrem treure el resultat en  $\Theta(n^2)$  passos addicionals canviant cada entrada diferent de 0 per 1 i deixant les que són 0 a 0. El temps total serà  $\Theta(n^{\log_2 7})$ .

**Proposta de solució al problema 4**

Per simplificar l'escriptura direm clica en lloc de subgraf complet i  $k$ -clica en lloc de subgraf complet de mida  $k$ .

L'algorisme  $B$  redueix CLICA a  $\text{CLICA}^{+1}$ .

Sigui  $\langle G, k \rangle$  un parell on  $G = (V, E)$  i  $k$  és un natural. Sigui  $\langle G', k \rangle$  el parell que retorna  $B$  amb entrada  $\langle G, k \rangle$ .  $B$  és de temps polinòmic, només ha d'afegir al graf d'entrada un nou vèrtex i  $n$  arestes.

Si  $G$  conté una  $k$ -clica aleshores  $G'$  conté una  $(k+1)$ -clica. Només cal afegir  $u_{\text{nou}}$  al conjunt de vèrtexs  $U$  que indueix una  $k$ -clica a  $G$ . Per la construcció de  $G'$ ,  $U \cup \{u_{\text{nou}}\}$  indueix un  $(k+1)$ -clica a  $G'$ .

Si  $G'$  conté una  $(k+1)$ -clica, volem demostrar que  $G$  conté una  $k$ -clica. Sigui  $U \subseteq V'$  el conjunt que induïx la clica a  $G'$  i  $|U| = k+1$ . Si  $u_{nou} \notin U$ , aleshores  $U \subseteq V$  induïx una  $(k+1)$ -clica a  $G$ . Si eliminem un vèrtex qualsevol  $v \in U$ , aleshores  $U - \{v\}$  induïx una  $k$ -clica a  $G$ . Si  $u_{nou} \in U$ , aleshores  $U - \{u_{nou}\}$  induïx una  $k$ -clica a  $G$ .

L'algorisme  $A$  redueix  $\text{CLICA}^{+1}$  a  $\text{CLICA}$ .

Sigui  $\langle G, k \rangle$  un parell on  $G = (V, E)$  i  $k$  és un natural. Sigui  $\langle G, k+1 \rangle$  el parell que retorna  $A$  amb entrada  $\langle G, k \rangle$ . És clar que  $A$  és de temps polinòmic; només ha d'incrementar la  $k$ .

Fixeu-vos que  $\langle G, k \rangle$  és un parell de  $\text{CLICA}^{+1}$  si i només si  $G$  conté una  $(k+1)$ -clica si i només si  $\langle G, k+1 \rangle$  és de  $\text{CLICA}$ .

### Proposta de solució al problema 5

- a) Es manté un diccionari que, per a cada una de les  $n$  paraules possibles mantingui un conjunt ordenat amb els identificadors dels documents que la contenen. En C++ es podria fer amb:

```
map<string, set<int>>> dic;
```

[Aquesta ED d'anomena un fitxer invertit i és semblant a l'índex de termes al final dels llibres.]

El pre-procés consistiria en, per a cada document  $d$ , per a cada paraula  $p$ , inserir  $d$  en  $\text{dic}[p]$ .

Per calcular la resposta a una consulta amb una sola paraula  $p$ , només cal iterar sobre tots els elements de  $\text{dic}[p]$ . Com que es troben en un conjunt ordenat, la resposta ja es donaria ordenada.

- b) Els diccionaris i els conjunts es poden guardar en espai lineal amb AVLs. Per tant, l'espai de l'ED seria  $O(N + n) = O(N)$ .
- c) El temps en el cas pitjor és  $O(N(\log n + \log M))$ , perquè per cadascuna de les  $N$  paraules, cal cercar-la en el diccionari que en té  $\leq n$ , i inserir-la en un conjunt que en té  $\leq M$ .
- d) En el cas pitjor el temps és  $O(\log n + D)$ . El primer terme és per cercar en el diccionari que té  $n$  elements. El segon terme és el temps per recórrer el seu conjunt de documents, que en té  $D$ .
- e) Per a cada paraula de la consulta, es pot obtenir la llista ordenada dels documents on apareix (com a la Part 1). Després, cal retornar la intersecció ordenada d'aquestes llistes. Hi ha moltes maneres de fer-ho, per exemple, fent  $k-1$  fusions.
- f) Cal repetir  $k$  cops el temps  $O(\log n + D)$  per cercar la llista per cada paraula. Això dona  $k$  llistes de  $\leq D$  elements cadascuna. Les  $k$  fusions tenen doncs cost  $O(kD)$ . En total, doncs,  $O(k(\log n + D))$ .

Nota: Es podria utilitzar també un

```
unordered_map<string, set<int>>> dic;
```

En aquest cas, els costos de temps en el cas mitjà s'obtindrien fent desaparèixer els factors logarítmics.

**Solució de l'Examen Final EDA****15/1/2014****Problema 1**

Aquest problema és el mateix que el primer problema del parcial.

**Problema 2**

(a)  $st + sr - a[k] \geq l$        $st + a[k] \leq u$        $sols(0,0,s,x)$

(b) Només cal invertir l'ordre de les crides recursives: és a dir, intercanviar els dos **ifs** (amb el seu codi corresponent) que hi ha dins de l'**else**.

**Problema 3**

Per mantenir l'ED requerida utilitzarem aquestes informacions:

```
unordered_map<string, int> urls;
```

```
vector<unordered_map<string, int>::iterator> iters;
```

El diccionari *urls* mantindrà les  $n$  URLs de les pàgines com a claus. Per a cada clau, es mantindrà com a informació un enter de 0 a  $n - 1$  (tots diferents) que correspon a la seva posició al vector *iters*. El vector *iters* mantindrà  $n$  iteradors: per a cada pàgina es tindrà un iterador que apunti a la seva posició en el diccionari URLs (no importa en quina posició). Caldrà vetllar que les dues estructures de dades sempre s'enllacin mútuament.

L'operació *insert\_url* hauria d'inserir la nova URL al diccionari, i col·locar l'iterador que apunta al nou element al final del vector. Això es fa en temps constant en mitjana.

L'operació *random\_url* hauria de triar un enter  $r$  a l'atzar entre 0 i  $n - 1$ , on  $n$  és la talla d'*urls* i d'*iters*. Després, accediria al  $r$ -èsim element de *iters* i retornaria la clau de l'element apuntat per aquell iterador. Això es fa en temps constant en el cas pitjor.

L'operació *remove\_url* hauria de cercar l'URL en el diccionari, trobar la seva posició al vector (gràcies a la informació associada), col·locar el darrer iterador del vector al forat del que s'esborra, actualitzar la informació del mogut al diccionari, i treure l'últim element del vector. Això es fa en temps constant en mitjana.

L'espai necessari per emmagatzemar  $n$  pàgines és  $\Theta(n)$ , a banda de l'espai necessari per les seves strings.

**Problema 4**

(a) La reducció crea un objecte nou  $x_{n+1}$  i una comanda  $C_{k+1} = (x_{n+1}, \neg x_{n+1})$  que s'afegeix a la llista de comandes. Evidentment aquesta reducció es pot computar en temps polinòmic. A més, si la instància del problema del Johnny té com a mínim una solució, llavors la instància resultant de la reducció en tindrà com a mínim dues: la original amb l'objecte  $x_{n+1}$  afegit a la solució, i la original sense l'objecte  $x_{n+1}$  afegit a la solució. Inversament, si la instància del problema del Johnny no té cap solució, llavors la instància resultant de la reducció tampoc en tindrà cap, i per tant no en tindrà almenys dues.

(b) El certificat és un vector de  $n$  bits, on l' $i$ -èssim bit indica si l'objecte  $x_i$  està a la solució o no. El verificador només ha de comprovar que, per cada comanda, o bé com a mínim un dels objectes desitjats per aquella comanda té el seu bit a 1 en el certificat, o bé com a mínim un dels objectes no desitjats per aquella comanda té el seu bit a 0 en el certificat. El certificat té mida  $n$  i per tant polinòmica en la mida de l'entrada, i el verificador pot executar la seva



tasca en temps polinòmica en la mida de l'entrada simplement recorrent cada clàusula i consultant el vector de bits.

**Problema 5**

(a) Un arbre d'expansió d'un graf és un subgraf que és connex i acíclic, i que conté tots els vèrtexs del graf. L'arbre d'expansió és mínim si la suma dels pesos de les seves arestes és mínima entre tots els arbres d'expansió.

(b) `not usat[v]`      `Q.push(WEdge(p, pair<int, int>(v, w)))`

## Solució de l'Examen Final EDA

20/6/2014

## Proposta de solució al problema 1

Aquest problema ja va sortir al parcial; no repetim la solució que ja vam publicar.

## Proposta de solució al problema 2

a) Primer fem una funció auxiliar que refà l'ABC el preordre del qual és el subvector  $v[e, \dots, d]$  (amb la preconditionió que  $v[e, \dots, d]$  és el preordre correcte d'algun ABC).

```

abc refer_i (const vector<Elem> &v, int e, int d) {
 if (d < e) return NULL;
 int i = e + 1;
 bool found = false;
 while (i ≤ d and not found) {
 found = (v[i] > v[e]);
 if (not found) ++i;
 }
 return new(v[e], refer_i(v, e + 1, i - 1), refer_i(v, i, d));
}

```

Amb aquesta auxiliar, el que volem és  $refer\_i(v, 0, v.size() - 1)$ .<sup>1</sup>

b) Primer plantejem una funció auxiliar que refà un AVL l'inordre del qual és el subvector  $v[e, \dots, d]$  (amb la preconditionió que  $v[e, \dots, d]$  està ordenat).

```

abc munta_AVL(const vector<Elem> &v, int e, int d) {
 if (d < e) return NULL;
 int m = (e + d)/2;
 return new node(v[m], munta_AVL(v, e, m - 1), munta_AVL(v, m + 1, d));
}

```

Amb aquesta auxiliar, el que volem és  $munta\_AVL(v, 0, v.size() - 1)$ .

c) Completem el codi:

```

bool es_AVL(abc T) {
 if (T == NULL) return true;
 else if (T->fe == NULL or T->fd == NULL) return (T->alc ≤ 1);
 else {
 if (T->fe->alc - T->fd->alc > 1) return false;
 if (T->fd->alc - T->fe->alc > 1) return false;
 if (not es_AVL(T->fe)) return false;
 if (not es_AVL(T->fd)) return false;
 return true;
 }
}

```

<sup>1</sup>Aquesta solució té cost  $\Theta(n^2)$  en el cas pitjor (p.ex. amb l'arbre xurro cap a l'esquerra). Hi ha una solució de cost  $\Theta(n)$  que és bastant més difícil de trobar i que no s'exigia.

**Proposta de solució al problema 3**

a)

```

bool es_torneig (const Graf& G) {
 int n = G.size ();
 for (int u = 0; u < n; ++u) {
 if (G[u][u]) return false;
 for (int v = u+1; v < n; ++v) {
 if (G[u][v] == G[v][u]) return false;
 }
 }
 return true;
}

```

b) Base: És clar que un graf torneig amb un vèrtex o dos vèrtexs té un camí Hamiltonià. Inducció: Suposem que tot graf torneig amb  $n$  vèrtexs té un camí Hamiltonià. Considerem un graf torneig  $G$  amb  $n + 1$  vèrtexs. Sigui  $u$  un vèrtex qualsevol de  $G$ . Llavors  $G - u$  és un graf torneig de  $n$  vèrtexs i, per hipòtesi d'inducció, té un camí Hamiltonià  $v_1, v_2, \dots, v_n$ . Si  $(u, v_1)$  és un arc de  $G$ , llavors  $u, v_1, v_2, \dots, v_n$  és un camí Hamiltonià de  $G$ . Sinó,  $(v_1, u)$  ha de ser un arc de  $G$  (perquè és torneig). Si  $(u, v_2)$  també és un arc de  $G$ , llavors  $v_1, u, v_2, \dots, v_n$  és un camí Hamiltonià de  $G$ . Sinó,  $(v_2, u)$  ha de ser un arc de  $G$  (perquè és torneig). Continuant així fins a considerar  $v_n$ , arribem a que si  $(u, v_n)$  no és un arc de  $G$  llavors  $(v_n, u)$  ho ha de ser (perquè és torneig), i llavors  $v_1, \dots, v_n, u$  és un camí Hamiltonià de  $G$ .

c)

```

list <int> cami (const Graf& G) {
 int n = G.size (); // suposarem $n \geq 2$, altrament poseu ifo.
 list <int> L;
 if (G[0][1]) L = {0, 1}; else L = {1, 0};
 for (int u = 2; u < n; ++u) insereix (L, u, G);
 return L;
}

```

```

void insereix (list <int>& L, int u, const Graf& G) {
 auto it1 = L.begin ();
 if (G[u][*it1]) { L.push_front (u); return; }
 auto it2 = it1; ++it2;
 while (it2 != L.end()) {
 if (G[*it1][u] and G[u][*it2]) { L.insert (it2, u); return; }
 ++it1; ++it2;
 }
 L.push_back(u);
}

```

d) El cost de `camí()` en el cas pitjor es dona quan a cada iteració cal recórrer tota la llista a `insereix()`. Com que a la iteració  $i$ , la llista té  $i$  elements, el cost és proporcional a  $\sum_{i=1}^n i = \Theta(n^2)$ .

**Proposta de solució al problema 4**

El que ha volgut dir la Steffy és que el graf proposat és un contraexemple a l'afirmació que el vèrtex  $v^{\text{MID}}$  no és necessari per obtenir una reducció correcta de DHAM a UHAM. I té

raó: si s'aplica aquesta transformació modificada al graf suggerit, el que queda és un graf no dirigit que conté un cycle Hamiltonià, p.ex. aquest:

$$e^{\text{OUT}}, a^{\text{IN}}, b^{\text{OUT}}, b^{\text{IN}}, c^{\text{OUT}}, c^{\text{IN}}, a^{\text{OUT}}, d^{\text{IN}}, d^{\text{OUT}}, e^{\text{IN}}$$

malgrat que el graf dirigit d'inici no en té cap.

### Proposta de solució al problema 5

a)  $\Theta(|V| \cdot |E|)$

b) La imprecisió en l'afirmació és que, si bé és veritat que l'algorisme de Bellman-Ford permet que el graf dirigit d'entrada tingui pesos negatius, el que no diu l'afirmació és que si el graf té algun cycle de pes negatiu accessible des del vèrtex de partida llavors l'algorisme no és capaç de trobar els camins de pes mínim (perquè, de fet, *no hi ha* pes mínim). Per tant, donat que el resultat d'assignar pes  $-1$  a cada arc podria provocar cycles negatius, l'aplicació de Bellman-Ford no està justificada.

**Solució de l'Examen Final EDA****16/1/2015****Proposta de solució al problema 1**

a) Veurem que la suma és  $O(n^4)$  i  $\Omega(n^4)$ . Primer,  $\sum_{i=0}^n i^3 \leq n \cdot n^3 = n^4 = O(n^4)$ . Segon,  $\sum_{i=0}^n i^3 \geq \frac{n}{2} \cdot \left(\frac{n}{2}\right)^3 = \Omega(n^4)$ . [Per ser més precisos: la desigualtat només és vàlida si  $n$  és parell. Però si  $n$  és senar llavors  $\sum_{i=0}^n i^3 = \sum_{i=0}^{n-1} i^3 + n^3 = \Omega((n-1)^4) + n^3 = \Omega(n^4)$ . En qualsevol cas obtenim que  $\sum_{i=0}^n i^3 = \Omega(n^4)$ .]

b)  $g$  creix més ràpid que  $f$ . Per justificar-ho veurem que  $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ . Sigui  $h(n) = \log_2(f(n)/g(n))$ . Simple manipulació dona  $h(n) = \sqrt{\log n} - \log_2 n$  i per tant  $\lim_{n \rightarrow +\infty} h(n) = -\infty$ . Això implica que  $\lim_{n \rightarrow +\infty} 2^{h(n)} = 0$  i per tant  $\lim_{n \rightarrow +\infty} f(n)/g(n) = 0$ . Pel criteri del límit en concloem que  $f(n) = O(g(n))$  i  $f(n) \neq \Omega(g(n))$ . Per tant  $g$  creix més ràpid que  $f$ .

**Proposta de solució al problema 2**

a)

```

int menors(abc T, const Clau& x) {
 if (T == NULL) return 0;
 if (x < T->k) return menors(T->esq, x);
 else {
 int aux;
 if (T->esq == NULL) aux = 0; else aux = (T->esq)->size;
 return 1 + aux + menors(T->dre, x);
 }
}

```

b)

```

// Precondició: T no conté k
void inserta(abc& T, const Clau& k, const Valor& v) {
 if (T == NULL) {
 T = new node;
 T->esq = T->dre = NULL;
 T->k = k; T->v = v;
 T->mida = 0;
 }
 else if (k < T->k) inserta(T->esq, k, v);
 else if (k > T->k) inserta(T->dre, k, v);
 else throw ERROR_PRECONDICIO;
 ++T->mida;
}

```

**Proposta de solució al problema 3**

a) Una possible solució és:

```

bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
 vector<bool>& marked, bool is_red) {
 col[u] = is_red;
 marked[u] = true;
 for (int v : g[u]) {

```

```

 if (not marked[v]) {
 if (not two_col_aux(g, v, col, marked, not is_red)) return false;
 }
 else if (col[v] == col[u]) return false;
}
return true;
}

```

b) El problema de 3-colorabilitat és NP-complet. Per tant, si el que diu el fòrum fos cert, tindríem un algorisme polinòmic per a resoldre un problema NP-complet, i per tant  $P = NP$ . Com que a dia d'avui el problema de si  $P = NP$  continua obert (i, a més, la conjectura predominant és que és fals), el que diu el fòrum no és creïble.

#### Proposta de solució al problema 4

a) Algorisme de Bellman–Ford.

b) Posem  $n = |V|$ . L'algorisme calcula la distància mínima entre un vèrtex inicial  $v$  i la resta de vèrtexs. Es manté un vector *dist* amb el millor cost trobat fins al moment des de  $v$  fins a cada vèrtex  $x$ . Inicialment, tenim  $dist[x] = \infty$  per a tot  $x$ , excepte que  $dist[v] = 0$ . Aquí  $\infty$  denota un valor especial més gran que el cost de qualsevol camí. És útil usar-lo per simplificar el codi. La part principal de l'algorisme consisteix a “relaxar” tots els arcs  $n - 1$  vegades. Relaxar un arc  $x \xrightarrow{c} y$  significa fer  $dist[y] = \min(dist[y], dist[x] + c)$ . Per relaxar tots els arcs, cal tractar tots els veïns de cada vèrtex, guardats en llistes d'adjacències. És obvi que els valors de *dist* no es poden incrementar a cap iteració. Si en una iteració no es produeix cap canvi, l'algorisme es pot aturar. Aquesta és una millora en general, encara que en el cas pitjor cal seguir fent  $n - 1$  iteracions.

c) Fins i tot amb la millora, el cost en el cas pitjor és  $\Theta(|V|(|V| + |E|))$ , perquè cada iteració costa  $\Theta(|V| + |E|)$ . [nota: algunes fonts diuen  $\Theta(|V||E|)$ , però això està assumint que  $|E| = \Omega(|V|)$ , la qual cosa no sempre és certa].

d) Si en alguna de les  $|V| - 1$  iteracions no ha millorat cap distància, segur que no hi ha cap cicle negatiu. Altrament, només cal relaxar una vegada més. Hi ha algun cicle negatiu si i només si alguna distància millora.

**Solució de l'Examen Final EDA****12/06/2015****Proposta de solució al problema 1**

- a) Aquest ja va sortir al parcial i la solució és la mateixa.  
 b) L'algorisme serveix per comprimir dades i per tant la correcta és la tercera.  
 c) El problema és en efecte NP-complet i per tant no pot existir cap algorisme de cost polinòmic que el resolgui a menys que  $P = NP$ ; per tant la resposta correcta és la segona.

**Proposta de solució al problema 2**

a)

```

Arbre interseccio (Arbre a1, Arbre a2) {
 if (not a1 or not a2) return nullptr;
 Node* p = new Node;
 p->esq = interseccio (a1->esq, a2->esq);
 p->dre = interseccio (a1->dre, a2->dre);
 return p;
}

```

- b) El cas pitjor és  $\Theta(n1 + n2)$  (i es dona quan els dos arbres són iguals, p.ex.).

**Proposta de solució al problema 3**

```

void backtrack(int k) {
 if (k == n) { solution_found = true; return; }
 for (int j = 0; j < n and not solution_found; ++j) {
 int cell_col = board[k][j].color;
 if (not onion_in_col[j] and not onion_in_region[cell_col]
 and not onion_in_neighborhood(k, j)) {
 col[k] = j;
 board[k][j].has_onion = true;
 onion_in_col[j] = true;
 onion_in_region[cell_col] = true;
 backtrack(k+1);
 if (not solution_found) {
 onion_in_region[cell_col] = false;
 onion_in_col[j] = false;
 board[k][j].has_onion = false;
 }
 }
 }
}
} } }

```

**Proposta de solució al problema 4**

Les EDs quedarien així:

```

struct Usuari {
 ... // dades personals
 string nom; // no hi ha dos usuaris amb el mateix nom
 unordered_set<string> seguits; // conjunt dels noms dels usuaris seguits
 list<Piulada> piulades; // piulades de l'usuari en ordre invers
};

struct Xarxa {
 unordered_map<string, Usuari> usuaris;
};

```

El tipus *Usuari* és una extensió del donat; també podríem haver creat un tipus nou que es digui *UsuariExtes*. Aquesta diferència és menor.

Mantindrem tots els usuaris en un diccionari on les claus són els *noms* d'usuari i les dades són les tuples dels *Usuaris* associats. D'aquesta forma, podrem accedir eficientment a totes les dades dels usuaris a través del seu nom. Com que no ens importa l'ordre, podem utilitzar una taula de hash per implementar el diccionari.

Per guardar les relacions de seguiment, estendrem la tupla *Usuari* amb un conjunt de strings que identifiquen els usuaris seguits. Com que no ens importa l'ordre, podem utilitzar una taula de hash per implementar el conjunt. Les operacions de *afegir\_seguiment* ( $x, u1, u2$ ) i *treure\_seguiment* ( $x, u1, u2$ ) s'implementen cercant  $u1$  al diccionari *usuaris* i inserint/esborrant  $u2$  al conjunt *seguits* corresponent. El temps d'aquestes operacions és doncs constant en mitjana.

Per guardar les piulades, també estendrem la tupla *Usuari* amb una llista de piulades, aquesta llista guardarà les piulades en ordre cronològic invers. L'operació *piular\_missatge* ( $x, u, m$ ) ha d'afegir el missatge  $m$  al davant de la llista de piulades de l'usuari  $u$  trobat al diccionari. El temps d'aquesta operació és doncs constant en mitjana.

Per implementar *mes\_recents* ( $x, u, k$ ) cal trobar primer la llista  $L$  de seguits de l'usuari  $u$  amb el diccionari. Després, conceptualment, cal concatenar totes les llistes de piulades dels usuaris en  $L$ , ordenar-les per hora d'emissió inversa i quedar-se amb les  $k$  primeres.

Però això es pot fer més eficient amb una cua de prioritats: Partint d'una cua de prioritats buida, s'insereix el primer missatge (el més recent doncs) de cada usuari seguit en  $L$ . Després, es repeteix  $k$  vegades el procés següent:

- S'extreu la piulada més recent de la cua de prioritats (i s'afegeix al final de la llista de piulades retornades),
- s'afegeix a la cua de prioritats el següent missatge del mateix usuari que el que s'ha tret.

Si  $u$  segueix  $m$  usuaris i  $n = \max\{k, m\}$ , el cost d'aquest procés és  $O(n \log n)$ , que sembla prou eficient donat que  $k$  és petit i  $n$  no és massa gran.



**Solució de l'Examen Final EDA**  
**Proposta de solució al problema 1**

07/01/2016

(a) Les respostes són:

|                                                                                     | Cert | Fals | Obert |
|-------------------------------------------------------------------------------------|------|------|-------|
| SOR és a la classe P                                                                | X    |      |       |
| SOR és a la classe NP                                                               | X    |      |       |
| SOR és NP-difícil                                                                   |      |      | X     |
| SAT és a la classe P                                                                |      |      | X     |
| SAT és a la classe NP                                                               | X    |      |       |
| SAT és NP-difícil                                                                   | X    |      |       |
| Es pot reduir polinòmicament SOR a COL                                              | X    |      |       |
| Es pot reduir polinòmicament COL a SOR                                              |      |      | X     |
| No es pot reduir polinòmicament SAT a SOR                                           |      |      | X     |
| Es pot reduir polinòmicament COL a SOR, i no es pot reduir polinòmicament SAT a SOR |      | X    |       |

(b) Si  $x$  és l'element  $v[i]$ , aleshores l'algorisme té cost  $\Theta(i)$ . Per tant el cost mig és:

$$\sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot \text{Cost}(x = v[i]) =$$

$$\sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot \Theta(i) =$$

$$\Theta(\sum_{0 \leq i < n} \text{Prob}(x = v[i]) \cdot i) =$$

$$\Theta(\sum_{0 \leq i < n} \frac{i}{n}) =$$

$$\Theta(\frac{1}{n}) \cdot \Theta(\sum_{0 \leq i < n} i) =$$

$$\Theta(\frac{1}{n}) \cdot \Theta(n^2) =$$

$$\Theta(n)$$

(c) Donat un graf dirigit  $G = (V, E)$  amb pesos, l'algorisme de Floyd-Warshall calcula a la vegada el cost del camí mínim de tot vèrtex  $u \in V$  a tot vèrtex  $v \in V$ . En el cas pitjor, el seu cost en temps és  $\Theta(|V|^3)$ , i en espai  $\Theta(|V|^2)$ .

**Proposta de solució al problema 2**

(a) 42 42 23 12 12 12

(b) A cada iteració escriu l'element més petit de  $V[0..i]$ .

(c) El cost de la primera iteració és constant. Respecte a la resta d'iteracions, el cost està dominat per la inserció en  $S$ . Si  $i > 0$ , a la  $i$ -èsima iteració, en què  $S$  té inicialment  $i$  elements, en el cas pitjor aquesta inserció té cost  $\Theta(\log i)$ . Si a cada iteració es dona aquest cost, aleshores el cost total és:

$$\Theta(1) + \sum_{i=1}^{n-1} \Theta(\log i) = \Theta(1) + \Theta(n \log n) = \Theta(n \log n),$$

donat que de la fórmula d'Stirling tenim que  $\sum_{i=1}^{n-1} \Theta(\log i) = \Theta(n \log n)$ .

(d) En cas que el vector tingui elements repetits, el codi escriu el mateix.

(e) Una possible solució:

```
int min = V[0];
for (int i = 1; i < n; ++i) {
 if (V[i] < min)
 min = V[i];
 cout << min << ' ';
}
```

Cada iteració del bucle només requereix operacions de cost constant. Com que es fan  $n - 1$  voltes, el cost del codi en funció de  $n$  és  $\Theta(n)$ , que és estrictament millor que  $\Theta(n \log n)$ .

### Proposta de solució al problema 3

Una possible solució al problema:

```
void top_sorts_rec (int k, const Graph& G, vector<int>& sol, vector<int>& indeg) {
 int n = sol.size ();
 if (k == n)
 print_solution (sol);
 else
 for (int x = 0; x < n; ++x)
 if (indeg[x] == 0) {
 indeg[x] = -1;
 for (int y : G[x]) --indeg[y];
 sol[k] = x;
 top_sorts_rec (k+1, G, sol, indeg);
 for (int y: G[x]) ++indeg[y];
 indeg[x] = 0;
 }
}

void top_sorts (const Graph& G, vector<int>& sol) {
 int n = G.size ();
 vector<int> indeg(n, 0);
 for (int x = 0; x < n; ++x)
 for (int y: G[x])
 ++indeg[y];
 top_sorts_rec (0, G, sol, indeg);
}
```

**Proposta de solució al problema 4**

- (a) Omplim la matriu de clausura, diguem-ne  $Gstar$ , de la manera següent. Des de cada vèrtex  $u$  del graf es fa una DFS o una BFS, usant  $Gstar[u]$  com a vector de marques booleans. D'aquesta manera, després de processar el vèrtex  $u$  tenim marcats a  $Gstar[u]$  aquells vèrtexs als quals es pot arribar des de  $u$ . Cal fer doncs  $n$  recorreguts, cadascun dels quals costa en el cas pitjor  $\Theta(n^2)$ . En total el cost en el cas pitjor és  $\Theta(n^3)$ .
- (b) Per inducció. Suposem que  $k = 0$ . Aleshores  $(I + G)^k = I$ . El resultat és cert perquè un camí buit només pot connectar un vèrtex amb sí mateix.

Considerem ara el cas que  $k > 0$ . Tenim que:

$$(I + G)_{uv}^k = \sum_{0 \leq w < n} (I + G)_{uw}^{k-1} \cdot (I + G)_{wv}$$

Suposem que hi ha un camí de  $u$  a  $v$  amb com a molt  $k$  arestes. Si aquest camí té com a molt  $k - 1$  arestes, aleshores per hipòtesi d'inducció  $(I + G)_{uv}^{k-1} > 0$ ; com que a més  $(I + G)_{vv} > 0$ , necessàriament tenim  $(I + G)_{uv}^k > 0$ . Si en canvi aquest camí té exactament  $k$  arestes, aleshores hi ha un vèrtex intermig  $w$  tal que hi ha un camí de  $u$  a  $w$  amb  $k - 1$  arestes, i una aresta de  $w$  a  $v$ . De nou per hipòtesi d'inducció, tenim  $(I + G)_{uw}^{k-1} > 0$ , i  $(I + G)_{wv} > 0$ . Per tant,  $(I + G)_{uv}^k > 0$  altra vegada.

Suposem ara que no hi ha camí de  $u$  a  $v$  amb com a molt  $k$  arestes. En particular, per tot  $w$  tal que  $0 \leq w < n$ , o bé no hi ha camí de  $u$  a  $w$  amb com a molt  $k - 1$  arestes, o bé no hi ha aresta de  $w$  a  $v$ ; per tant,  $(I + G)_{uw}^{k-1} = 0$  o  $(I + G)_{wv} = 0$ . De forma que  $(I + G)_{uv}^k = 0$ .

- (c) Hi ha un camí del graf de  $u$  a  $v$  si i només si hi ha un camí del graf de  $u$  a  $v$  amb com a molt  $n - 1$  arestes. Però per l'apartat b), hi ha un camí en el graf de  $u$  a  $v$  amb com a molt  $n - 1$  arestes si i només si  $(I + G)_{uv}^{n-1} \neq 0$ . Per tant, un algorisme consisteix en calcular  $(I + G)^{n-1}$  combinant l'algorisme d'exponenciació ràpida amb l'algorisme d'Strassen, i canviar en la matriu resultant tots els coeficients diferents de 0 per 1. El cost és el de  $\Theta(\log n)$  multiplicacions de matrius, cadascuna de les quals costa  $\Theta(n^{\log_2 7})$  (ja que les operacions aritmètiques amb enters tenen cost  $\Theta(1)$ ). Així doncs, el cost total és  $\Theta(n^{\log_2 7} \log n)$ , que és millor que  $\Theta(n^3)$ .

**Solució de l'Examen Final EDA**  
**Proposta de solució al problema 1**

08/06/2016

- (a) El teorema mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma  $T(n) = aT(n/b) + \Theta(n^k)$  amb  $a > 0$ ,  $b > 1$  i  $k \geq 0$ , llavors, fent  $\alpha = \log_b a$ ,

$$T(n) = \begin{cases} \Theta(n^k) & \text{si } \alpha < k, \\ \Theta(n^k \log n) & \text{si } \alpha = k, \\ \Theta(n^\alpha) & \text{si } \alpha > k. \end{cases}$$

- (b) 2

- (c) Les *priority\_queue*  $\langle T \rangle$  s'implementen amb max-heaps. Si existís una funció

$T\& \text{top}();$

aleshores un usuari de la classe podria modificar l'arrel del heap i trencar l'invariant que tot node és més gran o igual que els seus fills.

**Proposta de solució al problema 2**

- (a) Sí  
 (b) No (les fulles han de ser negres)  
 (c) No (els fills d'un node vermell han de ser negres)  
 (d) Sí  
 (e) No (cal que sigui un arbre binari de cerca)  
 (f) No (el nombre de nodes negres en un camí de l'arrel a una fulla ha de ser constant)

**Proposta de solució al problema 3**

- (a) Anomenem  $C(n)$  el cost de l'algorisme en el cas pitjor en funció de  $n$ . Llavors:

- Si  $s \geq \frac{1}{2}$ :  $C(n) = C(sn) + \Theta(1)$
- Si  $s \leq \frac{1}{2}$ :  $C(n) = C((1-s)n) + \Theta(1)$

- (b) El cas pitjor es dona, per exemple:

- si  $s \geq \frac{1}{2}$ , quan  $x < v[l]$ .
- si  $s \leq \frac{1}{2}$ , quan  $x > v[r]$ .

- (c) Podem resoldre les recurrències separatament, o bé observar que:

$$C(n) = C(\max(s, 1-s)n) + \Theta(1) = C\left(\frac{n}{\frac{1}{\max(s, 1-s)}}\right) + \Theta(1)$$

Com que  $0 < s < 1$ , tenim  $0 < \max(s, 1-s) < 1$ , i per tant  $\frac{1}{\max(s, 1-s)} > 1$ . Aplicant el teorema mestre de recurrències divisores tenim que  $\alpha = k = 0$ , i per tant el cost és  $\Theta(\log n)$  (independentment de  $s$ ).

**Proposta de solució al problema 4**

(a) Una possible solució:

```

#include <limits>

const int oo = numeric_limits<int>::max();

int cost_minim(const vector<vector<pair<int,int>>>& G, int x, int y) {
 vector<int> cost(G.size (), +oo);
 cost[x] = 0;
 deque<int> dq;
 dq.push_back(x);
 while (not dq.empty()) {
 int u = dq.front ();
 dq.pop_front ();
 if (u == y) return cost[y];
 for (auto p : G[u]) {
 int v = p.first ;
 int w = p.second;
 if (cost[v] > cost[u] + w) {
 cost[v] = cost[u] + w;
 if (w == 0) dq.push_front (v);
 else dq.push_back(v);
 } } }
 return -1;
}

```

(b) Per una banda hi ha un cost proporcional al nombre de vèrtexs (inicialització del vector de costos). Per una altra, el cost del bucle és proporcional al nombre d'arestes que se segueixen (sent doncs el cas pitjor quan es recorren totes les arestes). En suma, el cost és  $\Theta(|V| + |E|)$ .

(c)  $\Theta((|V| + |E|) \log |V|)$ **Proposta de solució al problema 5**

(a) Una possible solució:

```

bool ok(const vector<int>& s, const set<pair<int,int>>& D) {
 for (auto d : D)
 if (s[d.first] == s[d.second])
 return false;
 return true;
}

bool te_solucio (int k, vector<int>& s, const ent_DESIGUALTATS& e) {
 if (k == e.n) return ok(s, e.D);
 for (int v = e.l; v ≤ e.u; ++v) {
 s[k] = v;
 if (te_solucio (k+1, s, e)) return true;
 }
 return false;
}

```

```
}

```

```
bool te_solucio (const ent_DESIGUALTATS& e) {
 vector<int> s(e.n);
 return te_solucio (0, s, e);
}
```

- (b) Quan no hi ha solució, l'algorisme considera cadascuna de les  $(u - l + 1)^n$  possibles assignacions de les  $n$  variables als valors en  $[l, u]$ . Per cadascuna d'aquestes assignacions, es realitza un treball amb cost  $\Omega(1)$ . Així doncs, el cost en el cas pitjor és  $\Omega((u - l + 1)^n)$ .

- (c) Una possible solució:

```
ent_DESIGUALTATS reduccio(const ent_COLOREJAT& ec) {
 ent_DESIGUALTATS ed;
 ed.l = 1;
 ed.u = ec.c;
 ed.n = ec.G.size ();
 for (int u = 0; u < ec.G.size (); ++u)
 for (int v : ec.G[u])
 if (u < v)
 ed.D.insert ({u, v});
 return ed;
}
```

- (d) No. Per l'apartat anterior tenim que, com que COLOREJAT és NP-difícil, aleshores DESIGUALTATS també ho és. Així doncs, si hi hagués un algorisme polinòmic per resoldre DESIGUALTATS, tindríem  $P = NP$  i hauríem resolt un problema llargament obert en informàtica teòrica.

## Solució de l'Examen Final EDA

12/01/2017

## Proposta de solució al problema 1

- (a) Un graf de  $n$  vèrtexs té  $O(n^2)$  arestes.
- (b) Un graf connex de  $n$  vèrtexs té  $\Omega(n)$  arestes.
- (c) Un graf complet de  $n$  vèrtexs té  $\Omega(n^2)$  arestes.
- (d) Un min-heap de  $n$  vèrtexs té  $\Theta(n)$  fulles.
- (e) Un arbre binari de cerca de  $n$  vèrtexs té alçada  $\Omega(\log n)$ .
- (f) Un arbre binari de cerca de  $n$  vèrtexs té alçada  $O(n)$ .
- (g) Un arbre AVL de  $n$  vèrtexs té alçada  $\Omega(\log n)$ .
- (h) Un arbre AVL de  $n$  vèrtexs té alçada  $O(\log n)$ .

## Proposta de solució al problema 2

- (a) La cerca en amplada.
- (b) L'algorisme de Dijkstra.
- (c) L'algorisme de Bellman-Ford.
- (d) Cal que no hi hagi cap cicle amb pes negatiu al graf.
- (e) Per inducció sobre el nombre d'arcs del camí.

- **Cas base:** Si el camí no té cap arc, el vèrtex de sortida  $u$  és el mateix que el d'arribada  $v$ . En aquest cas tenim doncs que  $\omega_\pi(c) = \omega(c) = 0$ . Com que  $\omega(c) - \pi(u) + \pi(v) = 0$ , es compleix el que volíem.
- **Cas inductiu:** Suposem que el camí té  $k$  arcs, és a dir, és de la forma  $(u_0, u_1, \dots, u_k)$ , on  $u_0 = u$  i  $u_k = v$ . Com que  $u_1, \dots, u_k$  és un camí de  $u_1$  a  $u_k$  amb  $k - 1$  arcs, podem aplicar la hipòtesi d'inducció. Per tant  $\omega_\pi(u_1, \dots, u_k) = \omega(u_1, \dots, u_k) - \pi(u_1) + \pi(u_k)$ . Però

$$\begin{aligned}
 \omega_\pi(u_0, \dots, u_k) &= \omega_\pi(u_0, u_1) + \omega_\pi(u_1, \dots, u_k) \\
 &= \omega_\pi(u_0, u_1) + \omega(u_1, \dots, u_k) - \pi(u_1) + \pi(u_k) \\
 &= \omega(u_0, u_1) - \pi(u_0) + \pi(u_1) + \omega(u_1, \dots, u_k) - \pi(u_1) + \pi(u_k) \\
 &= \omega(u_0, u_1) - \pi(u_0) + \omega(u_1, \dots, u_k) + \pi(u_k) \\
 &= \omega(u_0, \dots, u_k) - \pi(u_0) + \pi(u_k)
 \end{aligned}$$

- (f) Si  $\pi$  és un potencial, aleshores els pesos reduïts  $\omega_\pi$  són no negatius. Per tant, podem aplicar l'algorisme de Dijkstra per calcular les distàncies amb pesos  $\omega_\pi$  des de  $s$  a tots els vèrtexs. Llavors podem calcular les distàncies amb pesos  $\omega$  usant la següent observació: si  $u, v \in V$  i  $c$  és el camí mínim amb pesos  $\omega$  de  $u$  a  $v$  (que existeix per hipòtesi), aleshores  $c$  és el camí mínim amb pesos  $\omega_\pi$  de  $u$  a  $v$  i  $\omega(c) = \omega_\pi(c) + \pi(u) - \pi(v)$ .

**Proposta de solució al problema 3**

- (a) Si  $M$  és una matriu  $n \times n$ , la funció *matrix\_mystery*(**const matrix&**  $M$ ) calcula  $M^{\sum_{i=1}^n i}$ , o equivalentment,  $M^{\frac{n(n+1)}{2}}$ .
- (b) El producte de dues matrius  $n \times n$  que calcula la funció *aux* costa temps  $\Theta(n^3)$ . Com que es fan  $\Theta(n)$  iteracions, i en cadascuna es fan dos productes de matrius amb cost  $\Theta(n^3)$ , en total el cost és  $\Theta(n^4)$ .
- (c) Una possible solució:

```
matrix exp(const matrix& M, int k) {
 if (k == 1) return M;
 matrix P = exp(M, k/2);
 if (k % 2 == 0) return aux(P, P);
 else return aux(aux(P, P), M);
}
```

```
matrix mystery(const matrix& M) {
 int n = M.size ();
 return exp(M, n*(n+1)/2);
}
```

L'exponenciació ràpida fa  $\Theta(\log(n(n+1)/2)) = \Theta(\log(n))$  productes de matrius, cadascun dels quals costa temps  $\Theta(n^3)$ . En total, el cost és  $\Theta(n^3 \log n)$ .

**Proposta de solució al problema 4**

- (a) El testimoni és  $p$ .
- (b) El codi del verificador es troba entre les línies 4 i 16.
- (c) Una possible solució:

```
bool ham2_rec(const vector<vector<int>>& G, int k, int u, vector<int>& next) {
 int n = G.size ();
 if (k == n)
 return find(G[u].begin (), G[u].end (), 0) != G[u].end();

 for (int v : G[u])
 if (next[v] == -1) {
 next[u] = v;
 if (ham2_rec(G, k+1, v, next)) return true;
 next[u] = -1;
 }
 return false;
}

bool ham2(const vector<vector<int>>& G) {
 int n = G.size ();
 vector<int> next(n, -1);
 return ham2_rec(G, 1, 0, next);
}
```

- (d) Es pot reemplaçar la crida



```
return find (G[u].begin (), G[u].end (), 0) \neq G[u].end();
```

per

```
return not G[u].empty() and G[u][0] == 0;
```

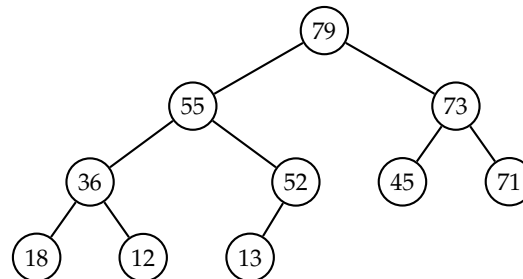
- (e) Si  $G$  és no connex llavors no pot ser Hamiltonià. En aquest cas només cal que la funció retorni **false**.

## Solució de l'Examen Final EDA

09/06/2017

## Proposta de solució al problema 1

(a) El max-heap resultant és:



(b) Per veure que  $P \subseteq \text{co-NP}$  cal veure que si  $L$  és un problema de la classe  $P$ , aleshores el seu complementari  $\bar{L}$  pertany a  $\text{NP}$ . Però si  $L$  és de  $P$  aleshores hi ha un algorisme polinòmic determinista  $A$  que decideix  $L$ . Considerem ara l'algorisme  $\bar{A}$  que fa el mateix que  $A$ , però retorna 1 quan  $A$  retorna 0, i retorna 0 quan  $A$  retorna 1. Llavors  $\bar{A}$  decideix  $\bar{L}$ , i com que  $\bar{A}$  triga temps polinòmic, tenim  $\bar{L} \in P \subseteq \text{NP}$ .

(c) El cost  $C(n)$  de  $f$  en funció de  $n$  segueix la recurrència

$$C(n) = 3C(n/3) + \Theta(1)$$

ja que hi ha 3 crides recursives sobre subvectors de mida  $n/3$  i addicionalment es fan operacions de cost constant. Pel Teorema Mestre de Recurrències Divisores, la solució a la recurrència és  $C(n) = \Theta(n)$ .

(d) No és certa. Per exemple,  $n^{2n} = (n^n)^2$  creix asimptòticament més ràpid que  $n^n$ .

## Proposta de solució al problema 2

Cal usar un diccionari amb claus enteres implementat amb una taula de hash, i un vector inicialment buit que contindrà la intersecció.

En primer lloc fem una passada sobre  $A$  i afegim tots els seus elements al diccionari. Es fan  $n$  insercions al diccionari, cadascuna de les quals triga temps  $O(1)$  en mitjana. De forma que la primera passada costa  $O(n)$  en mitjana.

En segon lloc fem una passada sobre  $B$  i, per a cadascun dels seus elements, mirem si ja pertany al diccionari. En cas afirmatiu, l'element s'afegeix al vector intersecció fent un *push\_back*. En cas contrari no es fa res. D'aquesta manera es fan  $m$  consultes al diccionari, cadascuna de les quals triga temps  $O(1)$  en mitjana. Com que cada *push\_back* triga temps constant, el cost de la segona passada és  $O(m)$  en mitjana.

En total, el cost és  $O(n + m)$  en mitjana.

## Proposta de solució al problema 3

(a) La taula plena és:

|          |   |   |   |   |   |   |   |   |               |   |
|----------|---|---|---|---|---|---|---|---|---------------|---|
| nivell : | A | B | C | D | E | F | G | H | profunditat : | 4 |
|          | 0 | 1 | 1 | 1 | 4 | 2 | 0 | 3 |               |   |

(b) La taula plena és:

|      | (1) | (2) | (3) | (4) |
|------|-----|-----|-----|-----|
| CERT |     | X   | X   |     |
| FALS | X   |     |     | X   |

(c) Una possible solució:

```

vector<int> levels(const vector<vector<int>>& G) {
 int n = G.size ();
 vector<int> lvl(n, -1), pred(n, 0);

 for (int u = 0; u < n; ++u)
 for (int v : G[u])
 ++pred[v];

 queue<int> Q;
 for (int u = 0; u < n; ++u)
 if (pred[u] == 0) {
 Q.push(u);
 lvl[u] = 0;
 }

 while (not Q.empty()) {
 int u = Q.front (); Q.pop();
 for (int v : G[u]) {
 --pred[v];
 lvl[v] = max(lvl[v], lvl[u]+1);
 if (pred[v] == 0) Q.push(v);
 }
 }
 return lvl;
}

```

La creació dels vectors té cost  $\Theta(n)$ . El primer bucle té cost  $\Theta(n + m)$ . El segon bucle té cost  $\Theta(n)$ . El tercer bucle té cost  $\Theta(n + m)$ . En total el cost és  $\Theta(n + m)$ .

#### Proposta de solució al problema 4

```

void write(const vector<int>& p, int n) {
 for (int k = 0; k < n; ++k) cout << " " << p[k];
 cout << endl;
}

void generate(int k, int n, vector<int>& p, vector<bool>& used) {
 if (k == n) write(p, n);
 else {
 for (int i = 0; i < n; ++i)
 if (not used[i] and k ≠ i) {
 used[i] = true;
 p[k] = i;
 generate(k+1, n, p, used);
 used[i] = false;
 }
 }
}

```

```
};
```

```
void generate_all (int n) {
 vector<int> p(n);
 vector<bool> used(n, false);
 generate (0, n, p, used);
}
```

## Solució de l'Examen Final EDA

19/01/2018

## Proposta de solució al problema 1

(a) Una possible solució:

```

void shift_down (vector<int>& v, int i) {
 int n = v.size()-1;
 int c = 2*i;
 if (c ≤ n) {
 if (c+1 ≤ n and v[c+1] < v[c]) c++;
 if (v[i] > v[c]) {
 swap(v[i], v[c]);
 shift_down(v, c);
 }
 }
}

```

(b) Una possible solució:

```

void update (vector<int>& v, int i, int x) {
 int y = v[i];
 v[i] = x;
 if (y < x) shift_down(v, i);
 else shift_up(v, i);
}

```

## Proposta de solució al problema 2

```

int n, k;
vector<int> perm;
vector<bool> used;

void write() {
 cout << perm[1];
 for (int i = 2; i ≤ n; ++i) cout << ' ' << perm[i];
 cout << endl;
}

void generate(int i, int lm, int mx) {
 if (lm > k or n - i + 1 + lm < k) return;
 if (i == n+1) write();
 else {
 for (int j = 1; j ≤ n; ++j)
 if (not used[j]) {
 used[j] = true;
 perm[i] = j;
 if (j > mx) generate(i+1, lm+1, j);
 else generate(i+1, lm, mx);
 used[j] = false;
 }
 }
}

```

```

int main() {
 cin >> n >> k;
 perm = vector<int>(n+1);
 used = vector<bool>(n+1, false);
 generate(1, 0, 0);
}

```

### Proposta de solució al problema 3

- (a) Donat un **string**  $s$  de mida  $n$ , la funció *mystery* calcula l'avaluació de  $s$  com a polinomi en  $x$ :  $\sum_{i=0}^{n-1} s[i] \cdot x^i$ .

El valor 0 no seria adequat per a  $x$  si es volgués fer servir *mystery* com a funció de hash perquè aleshores la funció retornaria sempre  $s[0]$ ; és a dir, tots els strings que comencen amb el mateix caràcter tindrien el mateix valor de hash.

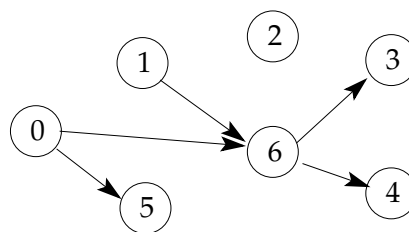
- (b) Les respostes:

|      | (1) | (2) | (3) | (4) |
|------|-----|-----|-----|-----|
| CERT |     | X   |     | X   |
| FALS | X   |     | X   |     |

- (c) El valor retornat en cridar *find*(2) és 0. El valor retornat en cridar *find*(9) és 5.
- (d) En el cas pitjor el valor buscat és estrictament més gran que tots els que hi ha al vector. D'aquesta manera a cada crida recursiva s'escull, de les dues parts, la més gran, que és  $\frac{2}{3}$  de la mida de l'interval  $[l, \dots, r]$ . A més, a banda de la crida recursiva només es fan operacions de cost constant. Així doncs, si  $n = r - l + 1$ , el cost  $T(n)$  en el cas pitjor ve determinat per la recurrència  $T(n) = T(\frac{2}{3}n) + \Theta(1)$ . Aplicant el teorema mestre de recurrències divisores, tenim que la solució de la recurrència és  $\Theta(\log n)$ .

### Proposta de solució al problema 4

- (a) La composició és:



- (b) Una possible solució:

```

matrix comp(const matrix& G1, const matrix& G2) {
 int n = G1.size();
 matrix C(n, vector<int>(n, false));
 for (int i = 0; i < n; ++i)
 for (int j = 0; j < n; ++j)
 for (int k = 0; k < n and not C[i][j]; ++k)
 C[i][j] = G1[i][k] and G2[k][j];
 return C;
}

```

En el cas pitjor (per exemple, quan els grafs  $G_1$  i  $G_2$  són buits) els tres bucles recorren els valors de 0 fins a  $|V| - 1$ . Com que cada volta del bucle més intern té cost constant, en total el cost en el cas pitjor és  $\Theta(|V|^3)$ .

- (c) Una forma alternativa d'obtenir la matriu d'adjacència  $C$  de la composició seria calcular el producte  $P$  de les matrius d'adjacència de  $G_1$  i  $G_2$  com a matrius de nombres enters, i aleshores fer  $C[i][j] = 1$  si i només si  $P[i][j] > 0$ .

Com que el producte de matrius es pot fer eficientment usant l'algorisme de Strassen en temps  $\Theta(|V|^{\log_2 7})$ , aquest seria el cost de l'algorisme resultant.

**Solució de l'Examen Final EDA**  
**Proposta de solució al problema 1**

20/06/2018

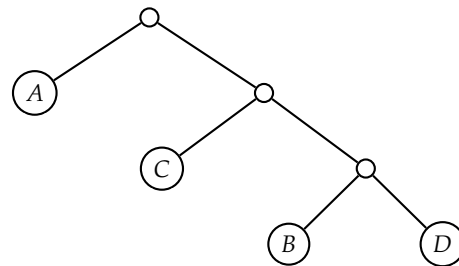
(a) Les *priority\_queue*  $\langle T \rangle$  s'implementen amb max-heaps. Si existís una funció

$T \& \text{top}();$

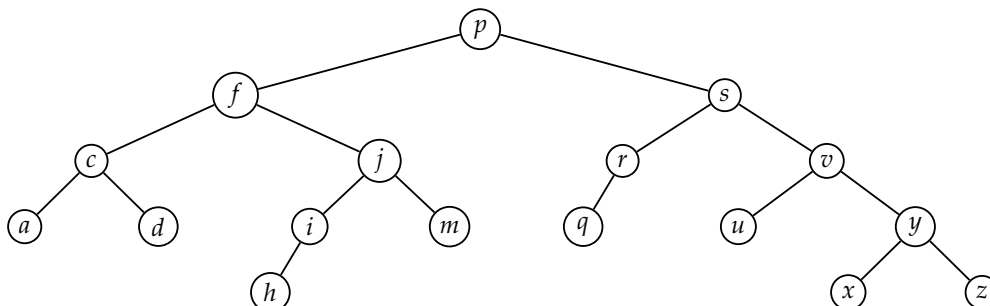
aleshores un usuari de la classe podria modificar l'arrel del heap i trencar l'invariant que tot node és més gran o igual que els seus fills.

(b)

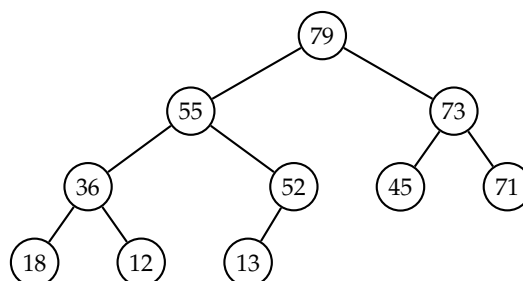
|   |     |
|---|-----|
| A | 0   |
| B | 110 |
| C | 10  |
| D | 111 |



(c)



(d)



**Proposta de solució al problema 2**

(a) Definim la funció  $U(m) = T(b^m)$ . Llavors  $T(n) = U(\log_b(n))$ . A més, tenim:

$$U(m) = T(b^m) = T(b^m/b) + \Theta(\log^k(b^m)) = T(b^{m-1}) + \Theta(m^k \log^k(b)) =$$



$$= T(b^{m-1}) + \Theta(m^k) = U(m-1) + \Theta(m^k)$$

El teorema mestre de recurrències substractores afirma que si tenim una recurrència de la forma  $U(m) = U(m-c) + \Theta(m^k)$  amb  $c > 0$  i  $k \geq 0$ , llavors  $U(m) = \Theta(m^{k+1})$ . Per tant la solució a la recurrència de l'enunciat és  $T(n) = \Theta((\log_b(n))^{k+1}) = \Theta(\log^{k+1} n)$ .

(b) Una possible solució:

```
bool search(const vector<int>& a, int x, int l, int r) {
 if (l == r) return x == a[l];
 int m = (l+r)/2;
 auto beg = a.begin();
 if (a[m] < a[m+1])
 return search(a, x, m+1, r) or binary_search(beg + l, beg + m + 1, x);
 else
 return search(a, x, l, m) or binary_search(beg + m+1, beg + r + 1, x);
}

bool search(const vector<int>& a, int x) {
 return search(a, x, 0, a.size()-1);
}
```

(c) El cas pitjor es dóna per exemple quan  $x$  no apareix a  $a$ . En aquesta situació el cost  $T(n)$  ve descrit per la recurrència  $T(n) = T(n/2) + \Theta(\log n)$ , perquè es fa una crida recursiva sobre un vector de mida  $\frac{n}{2}$  i el cost del treball no recursiu està dominat per la cerca binària, que té cost  $\Theta(\log(\frac{n}{2})) = \Theta(\log(n))$ . Aplicant el primer apartat tenim que la solució és  $T(n) = \Theta(\log^2(n))$ .

### Proposta de solució al problema 3

(a) Una possible solució és:

```
bool two_col_aux(const vector<vector<int>>& g, int u, vector<bool>& col,
 vector<bool>& marked, bool is_red) {
 col[u] = is_red;
 marked[u] = true;
 for (int v : g[u]) {
 if (not marked[v]) {
 if (not two_col_aux(g, v, col, marked, not is_red)) return false;
 }
 else if (col[v] == col[u]) return false;
 }
 return true;
}
```

(b) El problema de 3-colorabilitat és NP-complet. Per tant, si el que diu el fòrum fos cert, tindríem un algorisme polinòmic per a resoldre un problema NP-complet, i per tant  $P = NP$ . Com que a dia d'avui el problema de si  $P = NP$  continua obert (i, a més, la conjectura predominant és que és fals), el que diu el fòrum no és creïble.

**Proposta de solució al problema 4**

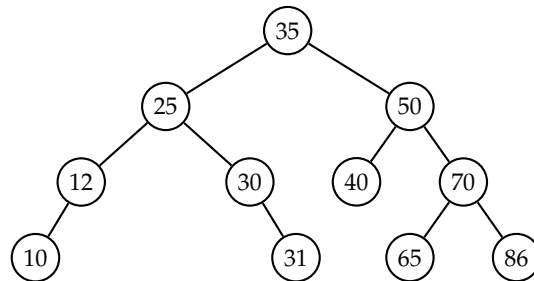
- (a)  $\boxed{sum\_chosen + sum\_rest - a[k] \geq l}$        $\boxed{sum\_chosen + a[k] \leq u}$        $\boxed{sols(0, 0, s)}$
- (b) Només cal invertir l'ordre de les crides recursives: és a dir, intercanviar els dos **ifs** (amb el seu codi corresponent) que hi ha dins de l'**else**.

## Solució de l'Examen Final EDA

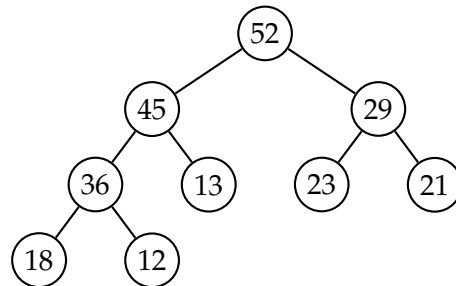
14/01/2019

## Proposta de solució al problema 1

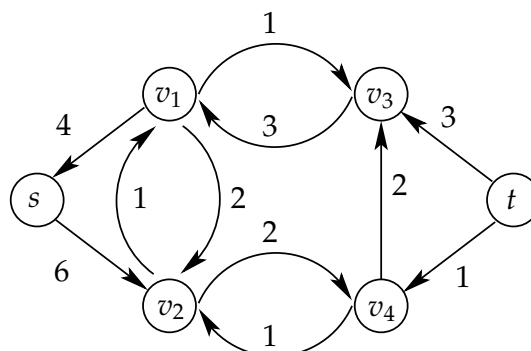
(a) L'arbre AVL resultant:



(b) El max-heap resultant:

(c)  $\Theta(n)$ (d)  $\Theta(\sqrt{n} \log n)$ (e)  $\Theta(\sqrt{n})$ 

(f) La xarxa residual:



## Proposta de solució al problema 2

(a) La funció *mystery* calcula el producte del polinomi  $p$  pel monomi  $c \cdot x^k$ . El seu cost en temps és  $\Theta(n + k)$ .

(b) El tipus `unordered_map<int,int>` es pot implementar amb taules de dispersió.

Una forma de completar el codi:

```

polynomial mystery(const polynomial& p, int c, int k) {
 polynomial q;
 for (auto mon : p)
 q[mon.first + k] = c * mon.second;
 return q;
}

```

El cost de la funció està dominat pel cost del bucle. Sigui  $m = p.size()$ . Es fan  $m$  voltes, a cadascuna de les quals l'operació més costosa és accedir a un parell del diccionari de  $q$ . Cada accés a aquest diccionari té cost  $O(1)$  en el cas mig. Així doncs el cost en el cas mig és com a molt  $O(m)$ .

(c) El cost de la funció és  $\Theta(n)$ .

(d) Una possible solució (similar a l'algorisme de Karatsuba):

```

polynomial operator*(const polynomial& p, const polynomial& q) {
 int n = p.size();
 if (n == 1) return polynomial(1, p[0]*q[0]);
 int n2 = n/2;

 polynomial p0(n2), p1(n2);
 for (int k = 0; k < n2; ++k) p0[k] = p[k];
 for (int k = n2; k < n; ++k) p1[k - n2] = p[k];

 polynomial q0(n2), q1(n2);
 for (int k = 0; k < n2; ++k) q0[k] = q[k];
 for (int k = n2; k < n; ++k) q1[k - n2] = q[k];

 polynomial p0_q0 = p0 * q0;
 polynomial p1_q1 = p1 * q1;
 polynomial p1_q0_plus_p0_q1 = ((p0 + p1) * (q0 + q1)) +
 mystery(p0_q0 + p1_q1, -1, 0);

 return p0_q0 +
 mystery(p1_q0_plus_p0_q1, 1, n2) +
 mystery(p1_q1, 1, n);
}

```

La funció fa 3 crides recursives sobre polinomis de mida  $\Theta(n/2)$ . A més, el treball no recursiu (sumes, crides a la funció *mystery*) té cost  $\Theta(n)$ . Si anomenem  $T(n)$  el cost de la funció en termes de  $n$ , aleshores obtenim la recurrència  $T(n) = 3T(n/2) + \Theta(n)$ . Aplicant el teorema mestre de recurrències divisores obtenim que el cost és  $T(n) = \Theta(n^{\log 3})$ , que és millor que  $\Theta(n^2)$ .

### Proposta de solució al problema 3

- (a) L'error d'eficiència consisteix en què cada vegada que escollim un nou subconjunt, tornem a recórrer tot el vector  $S$  sencer, sense considerar els subconjunts escollits prèviament. D'aquesta forma podem agafar un mateix subconjunt diverses vegades, o considerar diferents permutacions de la mateixa col·lecció de subconjunts.

Una possible manera d'arreglar el codi és afegir un paràmetre que sigui l'índex del primer subconjunt que hem de considerar en l'elecció següent:

```
bool set_cover_rec (int f, int K, int N, const vector<set<int>>& S,
 vector<int>& w, int c) {
 if (c == N) return true;
 if (K == 0) return false;
 for (int i = f; i < S.size (); ++i) {
 for (int x : S[i]) {
 if (w[x] == 0) ++c;
 ++w[x];
 }
 if (set_cover_rec (i+1, K-1, N, S, w, c)) return true;
 for (int x : S[i]) {
 --w[x];
 if (w[x] == 0) --c;
 }
 }
 return false;
}

bool set_cover (int K, int N, const vector<set<int>>& S) {
 vector<int> w(N, 0);
 return set_cover_rec (0, K, N, S, w, 0);
}
```

- (b) Anem a veure que la funció  $g$  defineix una reducció de VERTEX-COVER a SET-COVER. Clarament triga temps polinòmic en la mida de l'entrada de VERTEX-COVER. Falta veure que una entrada  $(K, G)$  de VERTEX-COVER és positiva si i només si  $g(K, G) = (K, N, S)$  també és positiva per a SET-COVER.

Si  $(K, G)$  és positiva per a VERTEX-COVER aleshores hi ha un recobriment per vèrtexs  $W \subseteq V$  amb  $|W| \leq K$ . Definim  $T = \{S_u \mid u \in W\}$ . Aleshores  $|T| \leq K$ . Sigui ara  $i$  tal que  $0 \leq i < N$ . Llavors l'aresta  $e_i = \{u, v\}$  compleix que  $i \in S_u, i \in S_v$ . A més, com que  $W$  és un recobriment,  $u \in W$  o  $v \in W$ . Si  $u \in W$  llavors  $S_u \in T$  i per tant tenim un subconjunt de  $T$  que inclou  $i$ . El cas  $v \in W$  és anàleg. Per tant,  $T$  és un recobriment per subconjunts de  $\{0, \dots, N-1\}$ .

Recíprocament, si  $(K, N, S)$  és positiva per a SET-COVER llavors hi ha un recobriment per subconjunts  $T \subseteq S$  de  $\{0, \dots, N-1\}$  amb  $|T| \leq K$ . Definim  $W = \{u \in V \mid S_u \in T\}$ . Llavors  $|W| \leq K$ . Sigui ara  $e_i \in E$  una aresta. Com que  $T$  és un recobriment, hi ha  $S_u \in T$  tal que  $i \in S_u$ . Per tant  $u$  és un extrem de  $e_i$  i a més  $u \in W$ . En conclusió,  $W$  és un recobriment per vèrtexs de  $G$ .

Per altra banda, la funció  $f$  no defineix una reducció correcta. Per exemple, suposem que  $K = 1$  i  $G$  és el graf següent:



Llavors l'entrada  $(K, G)$  és positiva per a VERTEX-COVER, perquè el vèrtex  $u_1$  cobreix

les dues arestes. Però  $f(K, G) = (1, 3, \{\{0, 1\}, \{1, 2\}\})$  no és positiva per a SET-COVER, perquè no hi ha cap conjunt que cobreixi tot sol  $\{0, 1, 2\}$ .

- (c) Si VERTEX-COVER es redueix polinòmicament a SET-COVER i a més tenim que VERTEX-COVER és NP-difícil (donat que és NP-complet), aleshores podem dir que SET-COVER és NP-difícil: donat un problema  $X$  de la classe NP, podem compondre la reducció de  $X$  a VERTEX-COVER amb la reducció de VERTEX-COVER a SET-COVER, i així obtenir una reducció de  $X$  a SET-COVER.

Però no podem deduir només amb això que SET-COVER sigui NP-complet, perquè falta justificar que pertany a NP. Anem a veure doncs que efectivament SET-COVER pertany a NP. En aquest cas els testimonis són la mida  $k$  i els  $k$  índexs dels subconjunts que formen el recobriment. Clarament els testimonis tenen mida no més gran que la de l'entrada. I verificar que una elecció de  $k$  subconjunts dóna un recobriment només requereix comprovar que cada nombre entre  $0$  i  $N - 1$  pertany a almenys un dels subconjunts. Això es pot fer en temps polinòmic en la mida de l'entrada ja que, com els subconjunts de l'entrada cobreixen  $\{0, \dots, N - 1\}$ , la mida de l'entrada és  $\geq N$ .

**Solució de l'Examen Final EDA****17/06/2019****Proposta de solució al problema 1**

- (a) Sigui  $S(n)$  la mida de la fórmula  $\text{xor}(x_1, \dots, x_n)$ . A partir de la definició, obtenim la recurrència  $S(n) = 4S(n/2) + \Theta(1)$ . Aplicant el teorema mestre de recurrències divisores, tenim que  $S(n) = \Theta(n^2)$ .
- (b) Sigui  $R$  la funció de l'enunciat. Vegem que  $R$  no és una reducció de  $k$ -COLOR a  $k+1$ -COLOR. Faltaria que es complís que, donat un graf  $G$ , si  $R(G)$  és  $k+1$ -colorable aleshores  $G$  és  $k$ -colorable. Però aquesta propietat no es compleix: per exemple, per a  $k=2$ , si  $G$  és un cicle de 3 vèrtexs, tenim que  $R(G) = G$  és 3-colorable, però  $G$  no és 2-colorable.
- (c) El problema de l'aparellament bipartit màxim.

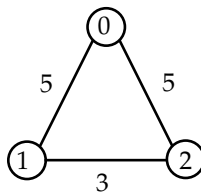
**Proposta de solució al problema 2**

- (a) Una possible solució:

$\text{Graph } g = \{ \{ \{3, 1\}, \{4, 2\} \}, \{ \{3, 0\}, \{5, 2\} \}, \{ \{4, 0\}, \{5, 1\} \} \};$

- (b) Al final de l'execució de la funció *mystery*, el valor de  $a[u]$  és la distància del vèrtex 0 al vèrtex  $u$ .
- (c) La funció *mystery* implementa l'algorisme de Dijkstra per trobar els camins mínims des del vèrtex 0 a tots els vèrtexs del graf  $g$ . Donat que el graf és connex, des de 0 s'arriba a cada vèrtex. Com que  $T$  consisteix en les arestes dels camins mínims des de 0, és un arbre d'expansió.

Però no és necessàriament un arbre d'expansió mínim. Per exemple, la funció *mystery* aplicada a aquest graf:



retorna les arestes  $\{ \{0, 1\}, \{0, 2\} \}$ . L'arbre corresponent té cost 10. Però l'arbre corresponent a les arestes  $\{ \{0, 1\}, \{1, 2\} \}$  també és un arbre d'expansió i té cost 8.

**Proposta de solució al problema 3**

Una possible solució:

```

node* pred(node* T, int x) {
 if (T == NULL) return NULL;
 else if (T->key > x) return pred(T->left, x);
 else {
 node* U = pred(T->right, x);
 if (U == NULL) return T;
 else return U;
 }
}

```

```

}

node* succ(node* T, int x) {
 if (T == NULL) return NULL;
 else if (T->key < x) return succ(T->right, x);
 else {
 node* U = succ(T->left, x);
 if (U == NULL) return T;
 else return U;
 }
}

pair<node*, node*> pred_succ(node* T, int x) {
 return {pred(T, x), succ(T, x)};
}

```

#### Proposta de solució al problema 4

(a) Una possible solució:

```

struct TSP {

 vector<vector<double>>> D;
 int n;
 vector<int> next, best_sol ;
 double best_tot_dist ;

 void recursive (int v, int t, double c) {
 if (t == n) {
 c += D[v][0];
 if (c < best_tot_dist) {
 best_tot_dist = c;
 best_sol = next;
 best_sol [v] = 0;
 }
 }
 else for (int u = 0; u < n; ++u)
 if (u != v and next[u] == -1) {
 next[v] = u;
 recursive (u, t+1, c+D[v][u]);
 next[v] = -1;
 }
 }

 TSP(const vector<vector<double>>>& D) {
 this->D = D;
 n = D.size ();
 next = vector<int>(n, -1);
 best_tot_dist = DBL_MAX;
 recursive (0, 1, 0);
 }
}

```



```
};
```

(b) Una possible solució:

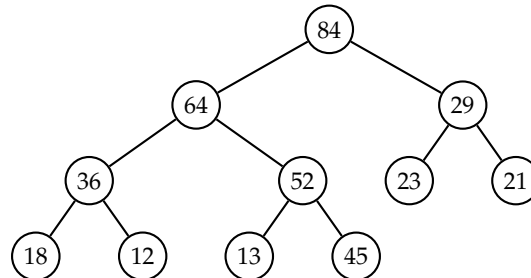
```
void recursive (int v, int t, double c) {
 if (c ≥ best_tot_dist) return;
 if (t == n) {
```

## Solució de l'Examen Final EDA

13/01/2020

## Proposta de solució al problema 1

(a) El heap resultant és:

(b)  $T(n) = 7 \cdot T(n/2) + \Theta(n^2)$ 

(c) Si ens n'adonem que la crida  $g(p)$  és constant respecte de  $n$ , és fàcil veure que la recurrència que descriu el cost de la funció  $f$  en funció de  $n$  és  $T(n) = T(n/2) + \Theta(1)$ . Per tant, el cost asimptòtic és  $\Theta(\log n)$ .

## Proposta de solució al problema 2

(a) La taula plena és:

|          |   |   |   |   |   |   |   |   |               |   |
|----------|---|---|---|---|---|---|---|---|---------------|---|
| nivell : | A | B | C | D | E | F | G | H | profunditat : | 4 |
|          | 0 | 1 | 1 | 1 | 4 | 2 | 0 | 3 |               |   |

(b) La taula plena és:

|      |     |     |     |     |
|------|-----|-----|-----|-----|
|      | (1) | (2) | (3) | (4) |
| CERT |     | X   | X   |     |
| FALS | X   |     |     | X   |

(c) Una possible solució:

```

vector<int> levels(const vector<vector<int>>& G) {
 int n = G.size ();
 vector<int> lvl(n, -1), pred(n, 0);

 for (int u = 0; u < n; ++u)
 for (int v : G[u])
 ++pred[v];

 queue<int> Q;
 for (int u = 0; u < n; ++u)
 if (pred[u] == 0) {
 Q.push(u);
 lvl[u] = 0;
 }

 while (not Q.empty()) {
 int u = Q.front (); Q.pop();
 for (int v : G[u]) {

```

```

--pred[v];
lvl[v] = max(lvl[v], lvl[u]+1);
if (pred[v] == 0) Q.push(v);
} }
return lvl;
}

```

La creació dels vectors té cost  $\Theta(n)$ . El primer bucle té cost  $\Theta(n + m)$ . El segon bucle té cost  $\Theta(n)$ . El tercer bucle té cost  $\Theta(n + m)$ . En total el cost és  $\Theta(n + m)$ .

### Proposta de solució al problema 3

```

(a) struct GasStations {
 int n;
 vector<int> cost;
 vector<vector<int>>> roads;
 vector<bool> best_solution;
 int best_cost;

 void rec(int i, vector<bool>& partial_sol, int partial_cost) {
 if (i == n) {
 best_cost = partial_cost;
 best_solution = partial_sol;
 }
 else {
 if (partial_cost + cost[i] < best_cost) {
 partial_sol[i] = true;
 rec(i+1, partial_sol, partial_cost + cost[i]);
 }

 bool needed = false;
 for (auto& c : roads[i])
 if (c < i and not partial_sol[c]) needed = true;

 if (not needed) {
 partial_sol[i] = false;
 rec(i+1, partial_sol, partial_cost);
 } } } // tanquem rec

```

- (b) Si ens demanem maximitzar el cost total d'instal·lació, la solució consistirà en col·locar una gasolinera a cada ciutat i el cost serà la suma de tots els costos del vector  $c$ . Per tant, implementaria aquesta idea de nou.

### Proposta de solució al problema 4

- (a) Una assignació que satisfà exactament un literal de cada clàusula és

$$I(x_1)=0, I(x_2)=1, I(x_3)=1, I(x_4)=1, I(x_5)=0$$

També n'hi ha una altra:

$$I(x_1)=0, I(x_2)=1, I(x_3)=1, I(x_4)=0, I(x_5)=1$$

No pot existir cap assignació que satisfaci exactament un literal de cada clàusula i compleixi  $I(x_1)=1$ . Si tenim  $I(x_1)=1$  aleshores utilitzant la primera clàusula veiem que necessàriament  $I(x_2)=1, I(x_3)=0$ . Gràcies a la segona clàusula, necessàriament  $I(x_4)=I(x_5)=0$ . Tot plegat, no tenim ja marge per satisfer la tercera clàusula.

Un conjunt de 3 clàusules de 3 literals que sigui una entrada positiva de **3-SAT** i negativa de **ONE-IN-THREE-SAT** és:

$$\begin{array}{ccccccc} x_1 & \vee & a & \vee & b \\ \neg x_1 & \vee & a & \vee & b \\ \neg x_1 & \vee & \neg a & \vee & \neg b \end{array}$$

L'assignació  $I(a)=1, I(b)=0, I(x_1)=0$  satisfà almenys un literal de cada clàusula.

Per veure que no podem satisfer exactament un literal de cada clàusula, vegem que si fem  $I(x_1)=1$  cert, aleshores necessàriament  $I(a)=I(b)=0$  gràcies a la primera clàusula, però això ja satisfà 2 literals a l'última clàusula. Si fem  $I(x_1)=0$  passa el mateix gràcies a la segona clàusula.

- (b) Sigui  $I$  una assignació que satisfà exactament un literal de cada clàusula de  $R(C)$ . Per reducció a l'absurd, assumim que no satisfà  $C$ , i per tant  $I(l_1)=I(l_2)=I(l_3)=0$ .

Gràcies a la primera clàusula de  $R(C)$ , necessàriament  $I(a)=I(b)=0$  i gràcies a la tercera clàusula, necessàriament  $I(c)=I(d)=0$ . Però aleshores  $I$  no satisfà la segona clàusula, el que és una contradicció.

Sigui  $I$  una assignació que satisfà almenys un literal de  $C$  i intentem construir una assignació  $I'$  que l'estengui i satisfaci exactament un literal de cada clàusula de  $R(C)$ .

Si  $I(l_2)=1$ , aleshores construïm  $I'(b)=I'(c)=0, I'(a)=I(l_1), I'(d)=I(l_3)$ .

En cas contrari  $I(l_2)=0$ .

- Si  $I(l_1)=1$ , aleshores construïm  $I'(a)=I'(c)=0, I'(b)=1, I'(d)=I(l_3)$ .
- Si  $I(l_1)=0$ , aleshores sabem que  $I(l_3)=1$  i podem construir  $I'(c)=1, I'(a)=I'(b)=I'(d)=0$ .

- (c) Per veure que **ONE-IN-THREE-SAT** és NP-difícil farem una reducció de **3-SAT** cap a ell. Donada una 3-CNF  $F$  amb clàusules  $\{C_1, C_2, \dots, C_m\}$ , construirem una 3-CNF  $F'$  amb clàusules  $\{R(C_1), R(C_2), \dots, R(C_m)\}$ . La 3-CNF  $F'$  té mida polinòmica respecte de  $F$ , ja que té  $3m$  clàusules i afegim  $4m$  variables noves, i es pot calcular en temps polinòmic.

Només ens falta veure que  $F$  és una entrada positiva de **3-SAT** si i només si  $F'$  és una entrada positiva de **ONE-IN-THREE-SAT**. Per això utilitzarem l'apartat anterior.

Si  $F \in \mathbf{3-SAT}$  aleshores prenem una assignació  $I$  que la satisfà i la podem estendre a una  $I'$  que satisfaci exactament un literal de cada clàusula de  $F'$ , pel que  $F' \in \mathbf{ONE-IN-THREE-SAT}$ . Això podem fer-ho perquè cada  $R(C_i)$  utilitza variables noves disjundes.

Si  $F' \in \mathbf{ONE-IN-THREE-SAT}$  aleshores qualsevol assignació que satisfà exactament un literal de cada clàusula de  $F'$  satisfà com a mínim un literal a cada clàusula de  $F$ , i per tant  $F \in \mathbf{3-SAT}$ .

- (d) Com que sabem que **ONE-IN-THREE-SAT** és NP-difícil, només cal veure que pertany a NP. En aquest cas, els candidats a testimonis seran assignacions, que tenen mida polinòmica respecte la fórmula. Comprovar un testimoni també és polinòmic, doncs només cal recórrer la fórmula comprovant que es satisfaci exactament un literal de cada clàusula. Finalment, només existiran testimonis vàlids per a les fórmules on pugui satisfer-se exactament un literal de cada clàusula.

Així doncs, com que **ONE-IN-THREE-SAT** és NP i NP-difícil, podem concloure que és NP-complet.

### Solució de l'Examen Final EDA Proposta de solució al problema 2A

09/06/2020

```

void recursive (){
 int u = s.back ();
 if (s.size () == n) {
 for (int v : G[u]) {
 if (v == 0){
 found = true;
 sol = s;
 } } }
 else {
 for (int v : G[u]) {
 if (not used[v]) {
 s.push_back(v);
 used[v] = true;
 recursive ();
 s.pop_back ();
 used[v] = false;
 if (found) return; } } } }

```

### Proposta de solució al problema 2B

```

void recursive (int c) {
 int u = s.back ();
 if (s.size () == n) {
 c += M[u][0];
 if (c < best_cost) {
 best_cost = c;
 best_sol = s;
 }
 }
 else {
 for (int v = 0; v < n; ++v)
 if (not used[v]) {
 if (c + M[u][v] < best_cost) {
 s.push_back(v);
 used[v] = true;
 recursive (c + M[u][v]);
 s.pop_back ();
 used[v] = false; } } } }

```

### Proposta de solució al problema 3A

- a) Ens fixem que  $G$  representa un graf dirigit, i per tant té sentit parlar de vèrtexs (peces) i arcs (parelles amb un ordre concret).

El bucle que calcula *indegree* recorre tot el graf (vèrtexs i arcs), i per tant té cost  $\Theta(n + m)$ . A continuació fem  $n$  insercions a la cua de prioritat, on cadascuna té cost  $O(\log n)$ .

Finalment, el bucle principal s'executa  $n$  vegades, ja que comença amb  $n$  elements a  $Q$  i elimina un element a cada iteració fins que la cua és buida. Així doncs tenim  $n$  crides a *remove\_min*, cadascuna amb cost  $O(\log n)$ . El bucle intern que crida a *decrease\_key* s'executa com a molt  $m$  vegades, una per cada arc. Per analitzar el cost de *decrease\_key* només cal veure que fa una crida a *find*, amb cost  $O(n)$  i fa surar un element en el heap, amb cost  $O(\log n)$ . Així doncs, el cost de les  $m$  crides és en total  $O(m \cdot n)$ .

Per tant, tenim un cost de  $O(n + m + n \log n + m \cdot n) = O(n \log n + m \cdot n)$ .

- b) La clau és fixar-se que el que ens demanen és un ordre topològic del graf  $G$ . I sabem que això es pot fer amb temps  $O(n + m)$ . Més concretament, comencen calculant *in\_degree* com en el codi proporcionat. A continuació, però, posem en un contenidor (pila, llista, cua, etc.) tots els vèrtexs de grau zero. Mentre el contenidor no sigui buit, traiem un element i decrementem en 1 l'*in\_degree* dels vèrtexs adjacents, tot afegint al contenidor els vèrtexs que passen a tenir *in\_degree* zero.
- c) Només cal afegir un nou mètode a la classe (funció comparació) i modificar 3 línies:

```
// CANVI (funció nova)
bool smaller (pair<Elem,Key>& p1, pair<Elem,Key>& p2) {
 return (p1.second < p2.second or
 (p1.second == p2.second and p1.first < p2.first));
}

void shift_up (int i) {
 if (i != 1 and smaller(v[i], v[i/2])) { // CANVI
 swap(v[i], v[i/2]);
 shift_up (i/2);
 }
}

void shift_down (int i) {
 int n = size ();
 int c = 2*i;
 if (c ≤ n) {
 if (c+1 ≤ n and smaller(v[c+1], v[c])) c++; // CANVI
 if (smaller (v[c], v[i])) { // CANVI
 swap(v[i], v[c]);
 shift_down (c);
 }
 }
}
```

### Proposta de solució al problema 3B

- a) El cost de la funció *find* és  $O(n)$ .

*Insercions.* El vèrtex  $x$  s'insereix a l'inici de la funció. En totes les altres insercions podem veure que sempre que s'insereix un vèrtex a  $Q$  és perquè  $Q$  no el conté i el vèrtex no ha estat tret de  $Q$  amb anterioritat (és a dir, mai ha estat dins  $Q$ ). Així doncs, cada vèrtex només es pot inserir una vegada a la cua.

*Extraccions.* Com que cada vèrtex l'afegim com a molt una vegada, és obvi que només el podrem treure com a molt una vegada.

*Decrements.* La funció *decrease\_key* la podem cridar quan eliminem un element  $u$  de la cua i explorem els arcs que surten de  $u$ . Hi pot haver, com a molt, una crida per cada arc que surti de  $u$ . Així doncs, en total podem cridar a *decrease\_key* com a molt  $m$  vegades.

*Cost.* Ens fixem que la funció *decrease\_key* primer fa una crida a *find*, amb cost  $O(n)$  i a continuació fa surar un element en el heap, amb cost  $O(\log n)$ . Per tant, té cost  $O(n)$ .

El cost total ve de les  $n$  crides a *remove\_min*, que impliquen temps  $O(n \log n)$ , les  $n$  crides a *insert*, que comporten temps  $O(n \log n)$  i les  $m$  crides a *decrease\_key*, amb temps  $O(m \cdot n)$ . Així doncs, el temps total és  $O(n \log n + m \cdot n)$ .

b) Cal afegir un atribut a la classe, reimplementar *find* i afegir 5 línies:

```
unordered_map<Elem,int> elem2idx; // CANVI: afegit atribut

void insert (const pair<Elem,Key> &x) {
 assert (not contains (x.first));
 v.push_back(x);
 elem2idx[x.first] = v.size () - 1; // CANVI
 shift_up (size ());
}

pair<Elem,Key> remove_min () {
 if (empty()) throw "Priority queue is empty";
 pair<Elem,Key> x = v[1];
 v[1] = v.back ();
 elem2idx[v[1].first] = 1; // CANVI
 v.pop_back ();
 shift_down (1);
 elem2idx.erase (x.first); // CANVI
 return x;
}

// CANVI: mètode reimplementat
int find (const Elem& x) {
 const auto& it = elem2idx.find(x);
 if (it == elem2idx.end()) return -1;
 else return it->second;
}

void shift_up (int i) {
 if (i != 1 and v[i/2].second > v[i].second) {
 swap(elem2idx[v[i].first],elem2idx[v[i/2].first]); // CANVI
 swap(v[i], v[i/2]);
 shift_up (i/2);
 }
}

void shift_down (int i) {
```



```

int n = size ();
int c = 2*i;
if (c ≤ n) {
 if (c+1 ≤ n and v[c+1].second < v[c].second) c++;
 if (v[i].second > v[c].second) {
 swap(elem2idx[v[i].first], elem2idx[v[c].first]); // CANVI
 swap(v[i], v[c]);
 shift_down(c);
 } } }

```

- c) Si repetim l'anàlisi de l'apartat a), només canvia que *decrease\_key* té ara cost  $O(\log n)$ . Per tant el cost total és  $O((n + m) \log n)$ .

Si  $m = \Theta(n^2)$ , el cost de la solució inicial és  $O(n \log n + n^2 \cdot n) = O(n^3)$ , mentre que amb la nova implementació de *find* té cost  $O((n + n^2) \log n) = O(n^2 \log n)$ . Per tant, sí que millora asimptòticament.

#### Proposta de solució al problema 4A

- a) És una entrada positiva perquè  $\{1, 2, 3, 4\}$  és un conjunt de mida 4 que compleix les propietats demanades.
- b) És una entrada positiva perquè  $\{2, 3, 5\}$  és un conjunt de mida 3 que compleix les propietats demanades.
- c) Per a demostrar que ENEMISTATS  $\in$  NP, em cal trobar un algorisme polinòmic no determinista.

- Testimonis: el conjunt de possibles testimonis estarà format per tots els subconjunts de  $T$  de mida  $r$ . És fàcil veure que la mida d'un testimoni és més petita que la mida de l'entrada.
- Verificador: el verificador rebrà  $(T, E, r)$  i un testimoni  $W$  i comprovarà, per totes les  $r(r-1)/2$  parelles  $u, v$  d'elements de  $W$ , que  $\{u, v\} \notin E$ . Cada comprovació té cost com a molt  $O(q)$ , i com que  $r \leq |T|$  tot plegat té cost polinòmic  $O(q \cdot |T|^2)$ .
- Només existeixen testimonis verificables per entrades positives: per definició, una entrada és positiva si i només si existeix  $M \subseteq T$  de mida  $r$  tal que tota parella d'elements de  $M$  no pertany a  $E$ . Aquest subconjunt  $M$  es correspon amb un testimoni, i per tant només existeixen testimonis verificables per les entrades positives.

- d) Donada una entrada de POBLES  $(P, C, k)$  construirem una entrada de ENEMISTATS  $(T, E, r)$  de la manera següent:

- $T = P$
- $E = \text{parelles}(P) \setminus C$ , on  $\text{parelles}(P) = \{\{u, v\} \mid u, v \in P \text{ i } u \neq v\}$ .
- $r = k$

Construir  $(T, E, r)$  a partir de  $(P, C, k)$  es pot fer en temps polinòmic. L'únic càlcul necessari és determinar el conjunt  $E$ , però això es pot fer considerant  $\text{parelles}(P)$ , que té mida quadràtica i per cada parella fer una cerca lineal a  $C$ .

També és fàcil veure que  $(T, E, r)$  té mida polinòmica respecte  $(P, C, k)$ , ja que només canviem  $E$ , que té mida com a molt quadràtica respecte la mida de  $P$ .

Finalment cal veure que  $(P, C, k) \in \text{POBLES} \iff (T, E, r) \in \text{ENEMISTATS}$ .

$\implies$ ) Si  $(P, C, k) \in \text{POBLES}$ , això vol dir que existeix  $S \subseteq P$  de mida  $k$  tal que per a tot  $u, v \in S$  amb  $u \neq v$  es compleix que  $\{u, v\} \in C$ . Aleshores si considerem  $M = S$ , ens n'adonem que  $M$  és un subconjunt de  $T$  de mida  $r$  (perquè  $r = k$ ), i que si prenem dos elements diferents  $u, v \in M$  qualssevol, es compleix que  $\{u, v\} \in C$  i per tant  $\{u, v\} \notin \text{parelles}(P) \setminus C = E$ . Per tant, l'existència de  $M$  demostra que  $(T, E, r)$  és una entrada positiva de ENEMISTATS.

$\impliedby$ ) Si  $(T, E, r) \in \text{ENEMISTATS}$ , és perquè existeix  $M \subseteq T$  de mida  $r$  tal que per a tot parell  $u, v \in M$  amb  $u \neq v$  es té que  $\{u, v\} \notin E$ . Aleshores considerem  $S = M$  i sabem que  $S$  és un subconjunt de  $P$  de mida  $k$  (perquè  $r = k$ ), i que si prenem una parella  $\{u, v\} \in S$  qualsevol es compleix que  $\{u, v\} \notin E = \text{parelles}(P) \setminus C$ , el que implica que  $\{u, v\} \in C$ . Per tant, l'existència de  $S$  demostra que  $(P, C, k)$  és una entrada positiva de POBLES.

- e) Per demostrar que és NP-complet ens cal veure que pertany a NP i és NP-difícil. Ho veiem de la manera següent. A l'apartat c) hem demostrat que ENEMISTATS  $\in$  NP. A l'apartat anterior hem vist que podem reduir POBLES a ENEMISTATS. Com que ara sabem que POBLES és NP-complet, i per tant NP-difícil, podem concloure que ENEMISTATS és NP-difícil.

#### Proposta de solució al problema 4B

- És una entrada positiva perquè  $\{1, 4, 5, 6\}$  és un conjunt de mida 4 que compleix les propietats demanades.
- És una entrada positiva perquè  $\{1, 2\}$  és un conjunt de mida 2 que compleix les propietats demanades.
- Per a demostrar que INVESTIGADORS  $\in$  NP, em cal trobar un algorisme polinòmic no determinista.
  - Testimonis: el conjunt de possibles testimonis estarà format per tots els subconjunts de  $I$  de mida  $r$ . És fàcil veure que la mida d'un testimoni és més petita que la mida de l'entrada.
  - Verificador: el verificador rebrà  $(I, M, r)$  i un testimoni  $W$  i comprovarà, per a totes les reunions  $\{u, v\} \in M$  (en tenim  $q$  en total), que  $u$  o  $v$  pertanyen a  $W$ . Cada comprovació es pot fer amb cost com a molt  $O(r)$  (mida de  $W$ ), i per tant en total el verificador trigarà  $O(q \cdot r)$ , que en ser  $r \leq |I|$  és polinòmic respecte la mida de  $(I, C, r)$ .
  - Només existeixen testimonis verificables per entrades positives: per definició, una entrada és positiva si i només si existeix  $S \subseteq I$  de mida  $r$  tal que per a tota reunió  $\{u, v\} \in M$ , almenys un dels dos elements pertany a  $S$ . Aquest subconjunt  $S$  es correspon amb un testimoni, i per tant només existeixen testimonis verificables per les entrades positives.
- Donada una entrada d'EQUIP  $(T, D, k)$  construirem una entrada d'INVESTIGADORS  $(I, M, r)$  de la manera següent:
  - $I = T$

- $M = \text{parelles}(T) \setminus D$ , on  $\text{parelles}(T) = \{\{u, v\} \mid u, v \in T \text{ i } u \neq v\}$ .
- $r = |T| - k$

Construir  $(I, M, r)$  a partir de  $(T, D, k)$  es pot fer en temps polinòmic. L'únic càlcul necessari és determinar el conjunt  $M$ , però això es pot fer considerant  $\text{parelles}(T)$ , que té mida quadràtica i per cada parella fer una cerca lineal a  $D$ .

També és fàcil veure que  $(I, M, r)$  té mida polinòmica respecte  $(T, D, k)$ , ja que només canviem  $M$ , que té mida com a molt quadràtica respecte la mida de  $T$ , i  $r$ , que també té mida polinòmica respecte  $(T, D, k)$ .

Finalment cal veure que  $(T, D, k) \in \text{EQUIP} \iff (I, M, r) \in \text{INVESTIGADORS}$ .

$\implies$ ) Si  $(T, D, k) \in \text{EQUIP}$ , això vol dir que existeix  $E \subseteq T$  de mida  $k$  tal que per a tot parell  $u, v \in E$  amb  $u \neq v$  es compleix que  $\{u, v\} \in D$ . Aleshores si considerem  $S = T \setminus E$ , ens n'adonem que  $S$  és un subconjunt de  $I$  de mida  $r = |T| - k$ , i que si prenem una parella  $\{u, v\} \in M$  aleshores  $u \in S$  o  $v \in S$ . Per veure aquest últim punt: si no fos cert, aleshores  $u \notin S$  i  $v \notin S$ , i com que  $S = T \setminus E$ , sabem que  $u \in E$  i  $v \in E$ . Gràcies a les propietats de  $E$ , sabem que  $\{u, v\} \in D$ , però això contradiu el fet que  $\{u, v\} \in M = \text{parelles}(T) \setminus D$ . Per tant, l'existència de  $S$  demostra que  $(I, M, r)$  és una entrada positiva de INVESTIGADORS.

$\impliedby$ ) Si  $(I, M, r) \in \text{INVESTIGADORS}$ , és perquè existeix  $S \subseteq I$  de mida  $r = |T| - k$  tal que per a tot parell  $\{u, v\} \in M$  es compleix  $u \in S$  o  $v \in S$  (o els dos). Aleshores si considerem  $E = T \setminus S$ , ens n'adonem que  $E$  és un subconjunt de  $T$  de mida  $|T| - (|T| - k) = k$  i que, a més, si prenem dos vèrtexs diferents  $u, v \in E$  es compleix que  $\{u, v\} \in D$ . Si no fos així, necessàriament hi hauria dos vèrtexs  $u, v \in E$  amb  $\{u, v\} \notin D$ . Per tant,  $\{u, v\} \in M$ . Però si  $u, v \in E = T \setminus S$  sabem que  $u, v \notin S$ . Aquest últim fet contradiu la propietat inicial del conjunt  $S$ . Per tant, l'existència d' $E$  demostra que  $(T, D, k)$  és una entrada positiva de EQUIP.

- e) Per demostrar que és NP-complet ens cal veure que pertany a NP i és NP-difícil. Ho veiem de la manera següent. A l'apartat c) hem demostrat que INVESTIGADORS  $\in$  NP. A l'apartat anterior hem vist que podem reduir EQUIP a INVESTIGADORS. Com que ara sabem que EQUIP és NP-complet, i per tant NP-difícil, podem concloure que INVESTIGADORS és NP-difícil.

## Solució de l'Examen Final EDA

08/01/2021

## Proposta de solució al problema 1

- (a) És fàcil veure que la funció visita cada element del vector una vegada. Per a cada element, la instrucció `++M[x]` busca l'element  $x$  al diccionari i l'incrementa. A continuació hi ha una altra cerca de  $x$  al diccionari. Per tant, el cost asimptòtic total és  $n$  vegades el cost d'una cerca.

Si utilitzem un AVL, aleshores cada cerca té cost en cas pitjor  $O(\log n)$ , pel que el cost total serà  $O(n \log n)$ .

Si utilitzem una taula de dispersió, cada cerca té cost en cas mitjà  $\Theta(1)$ , pel que el cost total serà  $\Theta(n)$ .

- (b) Ens hem de fixar en la funció recursiva *majority\_pairs* que pren dos arguments. Fins al primer bucle, totes les operacions són constants. El bucle té cost  $\Theta(n)$ , i crea un vector de mida com a molt  $n/2$ . La crida recursiva, doncs es fa sobre un vector de mida la meitat. A continuació, es fa una crida a la funció *times* sobre un vector de mida  $n$ . El cost d'aquesta funció, ja que el vector es passa per referència, es pot descriure com  $C(n) = C(n-1) + \Theta(1)$ , que té solució  $C(n) \in \Theta(n)$ . Tot plegat, veiem que el cost de la funció *majority\_pairs* es pot descriure amb la recurrència  $T(n) = T(n/2) + \Theta(n)$ . Aplicant el teorema mestre, podem afirmar que el cost en cas pitjor és  $\Theta(n)$ .

## Proposta de solució al problema 2

- (a) El codi resultant és:

```
void write_choices (vector<int>& partial_sol, int partial_sum, int idx) {
 if (partial_sum > money) return;
 if (idx == p.size ()) {
 if (partial_sum == money) {
 cout << "{";
 for (uint i = 0; i < partial_sol.size (); ++i)
 cout << (i == 0 ? "" : ",") << partial_sol[i];
 cout << "}" << endl;
 }
 }
 else {
 partial_sol.push_back(idx);
 write_choices (partial_sol, partial_sum + p[idx], idx+1);
 partial_sol.pop_back();
 write_choices (partial_sol, partial_sum, idx+1);
 }
}
```

- (b) En el codi anterior, podem la solució quan ja ens hem gastat més dels diners que tenim.

Adicionalment, podarem una solució parcial quan, fins i tot considerant que escollim tots els immobles que ens queden per processar, no podem arribar a la quantitat de diners que ens hem de gastar. És a dir, podem la cerca quan hem descartat massa immobles.

Per implementar-ho de manera eficient, passarem un paràmetre més al procediment *write\_choices*, que contindrà la suma dels preus dels immobles que ens queden per

processar. Dins el main, sumarem inicialment tots els preus i aquest serà el valor del paràmetre en la crida a *write\_choices*. En les dues crides recursives, el paràmetre es veurà decrementat en  $p[idx]$ . Finalment, si anomenem *remaining\_sum* a aquest paràmetre, després de la primera poda ja existent afegirem:

**if** (*partial\_sum* + *remaining\_sum* < *money*) **return**;

### Proposta de solució al problema 3

- (a) Sí, és una instància positiva. Per exemple, podem agrupar els nombres en 3 conjunts  $\{3, 8, 20\}, \{6\}, \{22\}$  de manera que no posem dos nombres que tinguin 1's a posicions comuns al mateix conjunt. Recordem que:

$$3 = 00011_2, 6 = 00110_2, 8 = 01000_2, 20 = 10100_2, 22 = 10110_2$$

- (b) Cal veure diverses coses:

- *Testimonis*: els candidats a testimonis són totes les possibles maneres que tenim de distribuir els elements de  $N$  en  $p$  conjunts. Observem que la mida d'un candidat és polinòmica respecte la mida de la instància. De fet, és lineal.
- *Verificador*: rebrà un parell  $(N, p)$  i  $p$  conjunts  $S_1, S_2, \dots, S_p$  on s'han distribuït els nombres de  $N$ . Per a cada conjunt  $S_i$ , considerarà totes les parelles de nombres de  $S_i$  (com a molt un nombre quadràtic de parelles a cada  $S_i$ ), i comprovarà que la representació en binari de la parella no tingui 1's en posicions comunes (això es pot fer en temps lineal en el nombre de bits del nombre més gran). Tot plegat es pot fer en temps polinòmic.
- *Testimonis només per instàncies positives*: una instància és positiva si i només si hi ha una manera de distribuir els nombres en subconjunts de manera correcta i aquesta distribució és precisament un testimoni. Per tant, les instàncies positives tenen testimonis, mentre que les instàncies negatives no en tindran.

- (c) La mida de  $(N, p)$  és polinòmica respecte la mida de  $(G, k)$ . D'una banda  $p = k$ . Pel que fa a  $N$ , aquest conté un nombre per cada vèrtex de  $G$ , i la mida d'aquests nombres coincideix amb el nombre d'arestes de  $G$ .

Anem a veure que  $(G, k) \in \text{COLORABILITAT} \Leftrightarrow (N, p) \in \text{DISTINCT-ONES}$ :

$(G, k) \in \text{COLORABILITAT} \Rightarrow (N, p) \in \text{DISTINCT-ONES}$ :

Si  $(G, k)$  és una instància positiva és perquè existeix una funció  $c$  que coloreja els vèrtexs amb  $k$  colors de manera que si dos vèrtexs tenen una aresta en comú aleshores tenen color diferent.

Podem distribuir els nombres  $x_i$  en  $p$  (que és igual a  $k$ ) conjunts de la manera següent:  $x_i$  va al conjunt  $S_j$  si i només si  $c(v_i) = j$  (és a dir, si  $v_i$  té color  $j$ ). Només ens cal veure que no posem al mateix conjunt dos nombres que tenen 1s en posicions comunes. Fem-ho per reducció a l'absurd: assumim que existeixen dos nombres  $x_r$  i  $x_s$  que van a un conjunt  $S_j$  i tenen un 1 a la posició  $i$ . Si tenen un 1 comú en el bit  $i$ -èssim és perquè els  $v_r$  i  $v_s$  pertanyen a l'aresta  $e_i$ . Si van al conjunt  $S_j$  és perquè  $c(v_r) = c(v_s) = j$ . I això no pot ser perquè els vèrtexs units per una aresta tenen colors diferents.

$(N, p) \in \text{DISTINCT-ONES} \Rightarrow (G, k) :$

Si  $(N, p)$  és una instància positiva és perquè podem agrupar els nombres de  $N$  en  $p$  conjunts  $S_1, S_2, \dots, S_p$  de manera que si dos nombres tenen 1s en posicions comunes aleshores van a conjunts diferents. Recordem, a més, que  $p = k$ .

Aleshores considerem la coloració següent pels vèrtexs de  $G$ :  $c(v_i) = j$  si i només si  $x_i$  pertany al conjunt  $S_j$ . Com que  $p = k$ , aquesta coloració utilitza com a molt  $k$  colors. A més, si prenem dos vèrtexs  $v_r$  i  $v_s$  que tenen una aresta  $e_i$  en comú, sabem per definició que  $x_r$  i  $x_s$  tindran el bit  $i$ -èsim a 1, i per tant no podran anar al mateix conjunt. Així doncs,  $c$  els assignarà un color diferent.

#### Proposta de solució al problema 4

- (a) És fàcil veure que el nombre de nodes  $N(k)$  d'un arbre binomial  $B_k$  es pot descriure amb la recurrència:

$$N(k) = \begin{cases} 1, & \text{si } k = 0 \\ 2 \cdot N(k-1), & \text{si } k > 0 \end{cases}$$

Podem demostrar per inducció sobre  $k$  que la solució a la recurrència és  $N(k) = 2^k$ . El cas base és trivial, ja que  $2^0 = 1$ . Assumim ara la hipòtesi d'inducció  $N(k-1) = 2^{k-1}$ . Per tant  $N(k) = 2 \cdot N(k-1) = 2 \cdot 2^{k-1} = 2^k$ , com volíem demostrar.

- (b) Com que el nombre de nodes d'un arbre binomial d'ordre  $k$  és  $2^k$ , i només hi pot haver com a molt un arbre binomial d'ordre  $k$  per a cada  $k$ , podem veure que en un heap binomial amb  $n$  nodes hi haurà un (i només un) arbre binomial d'ordre  $k$  si i només si el  $k$ -èsim bit menys significatiu d' $n$  en binari és 1.
- (c) Si l'arrel d' $A$  és menor que l'arrel de  $B$ , aleshores afegirem  $B$  com el fill de més a l'esquerra de l'arrel d' $A$ . Es cas contrari, afegirem  $A$  com el fill de més a l'esquerra de l'arrel de  $B$ .
- (d) El codi completat és el següent:

```
void BinomialHeap::merge(BinomialHeap& h){
// Make sure both have the same size (to make code simpler)
while (h.roots.size() < roots.size()) h.roots.push_back(NULL);
while (h.roots.size() > roots.size()) roots.push_back(NULL);
vector<Tree> newRoots(roots.size());
Tree carry = NULL;
for (int k = 0; k < roots.size(); ++k) {
 if (roots[k] == NULL and h.roots[k] == NULL) {
 newRoots[k] = carry;
 carry = NULL;
 }
 else if (roots[k] == NULL) {
 if (carry == NULL) newRoots[k] = h.roots[k];
 else {
 newRoots[k] = NULL;
 carry = mergeTreesEqualOrder(carry, h.roots[k]);
 }
 }
 else if (h.roots[k] == NULL) {
 if (carry == NULL) newRoots[k] = roots[k];
 else {
 newRoots[k] = NULL;
 carry = mergeTreesEqualOrder(roots[k], carry);
 }
 }
}
```

```
 newRoots[k] = NULL;
 carry = mergeTreesEqualOrder(carry, roots[k]);
 }
 else {
 newRoots[k] = carry;
 carry = mergeTreesEqualOrder(roots[k], h. roots[k]);
 }
}

if (carry != NULL) newRoots.push_back(carry);
roots = newRoots;
}
```

Podem apreciar que totes les operacions que s'hi fan són constants, pel que només hem de comptar quantes voltes dóna el bucle. El bucle fa tantes voltes com la mida de *roots*. La clau és adonar-se que aquest vector té tantes posicions com bits necessitem per representar  $n$ , i això són  $\Theta(\log n)$  posicions. Així doncs, el cost és  $\Theta(\log n)$ .

**Solució de l'Examen Final EDA****07/06/2021****Proposta de solució al problema 1**

- (a) Notem que  $n \log(n^2) = 2n \log n$ , i per tant, aquest funció té el mateix grau de creixement que  $n \log n$ . De fet, aquestes dues són les que tenen menor grau de creixement, seguides per  $n(\log n)^2$  i posteriorment per  $n^2$ .

Tot i que no se'ns demanava la justificació, l'adjuntem:

$$\lim_{x \rightarrow \infty} \frac{n \log n}{n(\log n)^2} = \lim_{x \rightarrow \infty} \frac{1}{\log n} = 0$$

Per tant,  $n \log n$  és menor asimptòticament parlant.

De forma similar:

$$\lim_{x \rightarrow \infty} \frac{n(\log n)^2}{n^2} = \lim_{x \rightarrow \infty} \frac{(\log n)^2}{n} = 0$$

- (b) Sabem que, o bé tots els problemes NP-complets pertanyen a  $P$  o cap d'ells hi pertany. No obstant, no se sap quina de les dues situacions és la correcta. Per tant, no sabem si l'afirmació que ens fan és certa o falsa.

Si fos certa, necessàriament tots els problemes NP-complets pertanyerien a  $P$ . Si fos falsa, cap d'ells ho faria.

- (c) Ens fixem que quan no estem al cas base el primer bucle s'executa  $a$  vegades, on a cada volta es fa una crida recursiva de mida la meitat. En acabar el primer bucle, passem a fer un treball (els dos bucles ennierats) que té cost  $\sum_{i=1}^{n-1} \Theta(i)$ , que equival a  $\Theta(n^2/2)$  i per tant  $\Theta(n^2)$ . Així doncs, el cost d'aquesta funció recursiva es pot descriure per:

$$T(n) = a \cdot T(n/2) + \Theta(n^2)$$

d'on identifiquem  $b = 2$ ,  $k = 2$  i  $\alpha = \log_2 a$ . Aplicant el teorema mestre de recurrències divisores podem afirmar que

- Si  $a < 4$ , aleshores  $\alpha < k$  i per tant,  $T(n) \in \Theta(n^2)$
- Si  $a = 4$ , aleshores  $\alpha = k$  i per tant,  $T(n) \in \Theta(n^2 \log n)$
- Si  $a > 4$ , aleshores  $\alpha > k$ , i per tant,  $T(n) \in \Theta(n^{\log_2 a})$



**Proposta de solució al problema 2**

```
(a) bool is_stable (const signed_graph& G, int u, vector<int>& team) {
 for (auto& e : G[u]) {
 int e_t = expected_team(team[u], e.sign);
 if (team[e.target] != no_team and team[e.target] != e_t) return false;
 if (team[e.target] == no_team) {
 team[e.target] = e_t;
 if (not is_stable (G, e.target, team)) return false;
 }
 }
 return true;
}
```

- (b) Ho podem fer perquè si existeix una manera correcta de dividir els vèrtexs en equips  $A$  i  $B$ , aleshores segur que existeix una manera correcta on el vèrtex 0 va a l'equip  $A$ . Només cal adonar-se que, donada una solució on 0 va a l'equip  $B$ , si canviem l'equip de tots els vèrtexs, obtindrem una solució correcta on 0 va a l'equip  $A$ .

**Proposta de solució al problema 3**

- (a) Dissenyarem una funció recursiva `canals(int l, int r)` que escriurà un conjunt vàlid de canals de transmissió per als treballadors  $\{l, l+1, \dots, r-1, r\}$ . Si  $l = r$  només hi ha un treballador i per tant no cal cap canal.

En cas contrari, prenem  $m$  el punt mig entre  $l$  i  $r$ , i considerem  $n = r - l + 1$ . Això ens parteix els treballadors en dues meitats d'aproximadament  $n/2$  elements. Primer escriurem, amb una crida recursiva, tots els canals necessaris per comunicar els treballadors  $\{l, l+1, \dots, m-1\}$  i a continuació, amb una altra crida recursiva, els canals necessaris per comunicar els treballadors  $\{m+1, m+2, \dots, r\}$ . Fixem-nos que les úniques comunicacions que ens falten són d'entre treballadors de meitats diferents (i també d'entre treballadors de l'esquerra cap a  $m$ ). Per tal de comunicar les dues meitats, crearem els canals següents:

- $l \rightarrow m, l+1 \rightarrow m, \dots, m-1 \rightarrow m$
- $m \rightarrow m+1, m \rightarrow m+2, \dots, m \rightarrow r-1, m \rightarrow r$

Intuïtivament,  $m$  és l'intermediari per les comunicacions entre treballadors de l'esquerra i treballadors de la dreta. Notem, a més, que hem creat exactament  $n - 1$  canals addicionals.

Adjuntem codi per més concreció, tot i que no es demanava.

```
void canals (int l, int r) {
 if (l >= r) return;
 else {
 int m = (l+r)/2;
 canals (l, m-1);
 canals (m+1, r);
 for (int k = l; k < m; ++k)
 cout << k << " --> " << m << endl;
 for (int k = m+1; k <= r; ++k)
 cout << m << " --> " << k << endl;
 }
}
```

```

}

int main() {
 int n; cin >> n;
 canals(1, n);
}

```

- (b) Fixem-nos que el nombre de canals que crea la nostra funció es pot descriure per la recurrència:

$$C(n) = 2C(n/2) + \Theta(n)$$

que, pel teorema mestre de recurrències divisores té solució  $C(n) \in \Theta(n \log n)$ .

#### Proposta de solució al problema 4

- (a) Construïrem una instància de 2SAT que serà la conjunció de les següents clàusules:

- Per a cada  $i \in V$ , una clàusula  $p_i$ .
- Per a cada  $i \in N$ , una clàusula  $\neg p_i$ .
- Per a cada  $(i, j) \in I$ , una clàusula  $\neg p_i \vee \neg p_j$ .
- Per a cada  $(i, j) \in R$ , una clàusula  $\neg p_i \vee p_j$ .

- (b) Com que hem construït una reducció polinòmica del problema dels equipaments cap a 2SAT, i sabem que 2SAT és un problema que es pot resoldre en temps polinòmic, aleshores podem concloure que el problema dels equipaments també es pot resoldre en temps polinòmic.

Tot i que no es demanava, remarquem que l'algorisme polinòmic pot consistir en calcular primer la reducció de l'apartat anterior i aplicar un algorisme polinòmic per 2SAT sobre la fórmula resultant.

**Solució de l'Examen Final EDA****10/01/2022****Proposta de solució al problema 1**

(a) El codi escriurà les 5 paraules següents:

pep, pe, ep, e, p

Pel que fa al cost, sabem que el cost de *mystery* ve donat per la recurrència  $T(n) = 2T(n-1) + \Theta(n)$ . Això és així perquè es fan dues crides recursives de mida  $n-1$  i les altres operacions tenen cost constant, excepte les crides a *substr*, que tenen cost  $\Theta(n-1) = \Theta(n)$ . Aquesta recurrència té solució  $T(n) \in 2^n$ .

(b) Sigui  $n = s.size()$ . Anem a comptar quantes vegades s'executa l'*insert* de dins el bucle. Per cada  $i$ , el bucle intern dona exactament  $n-i$  voltes. Com que  $i$  varia entre 0 i  $n-1$ , això ens dóna  $n + (n-1) + \dots + 1 = \Theta(n^2)$  crides a *substr*. Però amb això no en fem prou perquè cada crida a *substr* té cost  $j-i+1$ .

Anem a comptar el cost de totes aquestes crides. Recordem que per unes  $i, j$  concretes el cost de la crida a *substr* és  $j-i+1$ . Per una  $i$  concreta,  $j$  es mou entre  $i$  i  $n-1$  i per tant, el cost de les crides a *substr* amb aquesta  $i$  és  $1 + 2 + \dots + (n-i) = \Theta((n-i)^2)$ . Si sumem sobre totes les  $i$ 's el cost total és  $\sum_{i=0}^n \Theta((n-i)^2) = \Theta(n^3)$ .

(c) Una possible solució és:

```
void mystery(const string& s, unordered_set<string>& res){
 for (int k = 1; k ≤ s.size (); ++k)
 for (int i = 0; i + k ≤ s.size (); ++i)
 res . insert (s . substr (i ,k));
}
```

**Proposta de solució al problema 2**

(a) Una possible solució és:

```
bool find_ranking (vector<int>& ranking, vector<int>& pos_in_ranking,
 vector<bool>& used, int idx){
 if (idx == n) return good_ranking(pos_in_ranking);
 else {
 for (int i = 0; i < n; ++i) {
 if (not used[i]) {
 ranking[idx] = i;
 pos_in_ranking[i] = idx;
 used[i] = true;
 if (find_ranking(ranking, pos_in_ranking, used, idx+1)) return true;
 used[i] = false;
 } } }
 return false;
 }

bool good_ranking (const vector<int>& pos_in_ranking) {
 for (Match& g : matches) {
 if (pos_in_ranking[g.first] > pos_in_ranking[g.second]) return false;
 if (pos_in_ranking[g.second] > pos_in_ranking[g.third]) return false;
 }
}
```

```

 }
 return true;
}

```

- (b) Essencialment el codi genera totes les permutacions dels  $n$  jugadors i comprova si n'hi ha alguna de correcta. En cas pitjor no n'hi haurà cap de correcta i per tant haurà de generar i comprovar totes les permutacions. Així doncs es faran  $n!$  crides a *good\_ranking*.

Com que cada crida a *good\_ranking* té cost en cas pitjor  $\Theta(m)$ , això ens dona una fita inferior del codi de  $\Theta(m \cdot n!)$ .

- (c) Es pot solucionar el problema en temps polinòmic en  $n$  i  $m$ . Per a fer-ho construïm un graf dirigit on els nodes són els jugadors. Per cada partida amb resultat  $(j_1, j_2, j_3)$  afegim dos arcs  $j_1 \rightarrow j_2$  i  $j_2 \rightarrow j_3$ . Per tant afegirem com a molt  $\Theta(m)$  arcs. Un cop hem construït el graf buscarem una ordenació topològica en temps  $O(n + m)$ . Si l'ordenació topològica acaba sense haver afegit tots els jugadors voldrà dir que no hi ha un rànquing possible.

Nota: si no anem amb compte podríem afegir arcs repetits, però això no és un problema per a la correcció o el cost de l'algorisme de cerca topològica explicat a classe.

### Proposta de solució al problema 3

- (a) Anomenem  $X$  al problema d'aquest apartat. No és raonable pensar que podem solucionar  $X$  en temps polinòmic. Vegem per què. Considerem 5-COL el problema del 5-colorejat de grafs, que sabem que és NP-complet. Existeix una reducció de 5-COL cap a  $X$ : donat un graf  $G$  amb vèrtexs  $V$  i arestes  $E$ , considerem el conjunt de col·laboradors  $\{c_u | u \in V\}$ , i per cada aresta  $\{u, v\} \in E$  imposem que el col·laborador  $c_u$  vol evitar el col·laborador  $c_v$ . Com que hem reduït polinòmicament 5-COL a  $X$ , si  $X \in P$  també tindrem 5-COL  $\in P$ , i això és un problema obert a dia d'avui.
- (b) Anomenem  $Y$  al problema d'aquest apartat. Podem afirmar que  $Y \in P$ . Vegem per què. Considerem 2-COL el problema del 2-colorejat de grafs, que sabem que pertany a  $P$ . Existeix una reducció de  $Y$  cap a 2-COL: donat un conjunt de col·laboradors  $\mathcal{C}$  i una llista d'incompatibilitats  $I_c$  per cada  $c \in \mathcal{C}$ , construïm una instància de 2-COL que consisteix en el graf  $G$  amb vèrtexs  $\{v_c | c \in \mathcal{C}\}$  i arestes  $\{\{v_c, v_d\} | d \in \mathcal{C}, c \in I_d\}$ . Com que hem trobat una reducció de  $Y$  cap a 2-COL i aquest últim pertany a  $P$ , aleshores  $Y \in P$ .
- (c) Anomenem  $Z$  al problema d'aquest apartat. No és raonable pensar que podem solucionar  $Z$  en temps polinòmic. Vegem per què. Considerem PARTICIÓ, el problema de determinar si podem partir un conjunt d'enters en dues parts que sumin igual. Sabem que aquest problema és NP-complet. Existeix una reducció de PARTICIÓ cap a  $Z$ : donat un conjunt d'enters  $S$ , considerem el multiconjunt de col·laboradors  $\{c_s | s \in S\}$ , tal que el patrimoni de  $c_s$  és precisament  $s$ . Com que hem reduït polinòmicament PARTICIÓ a  $Z$ , si  $Z \in P$  també tindrem PARTICIÓ  $\in P$ , i això és un problema obert a dia d'avui.

### Proposta de solució al problema 4

- (a) Una possible solució és:

```

int search(int x, const vector<int>& v) {
 int n = v.size();
 if (n == 0) return -1;
 int b = 1;
 while (b < n and v[b] < x) b *= 2;
}

```

```
 return bin_search(x, v, b/2, min(n-1, b));
}
```

- (b) El primer que cal fer és observar el comportament del bucle. Després de  $k$  voltes, el valor de  $b$  és  $2^k$ . Fixem-nos que s'atura tan bon punt troba un valor  $b$  tal que  $v[b] \geq x$ . Per tant, s'atura amb un nombre de voltes  $k$  tal que  $v[2^k] \geq x$  però tal que  $v[2^{k-1}] < x$  i això ens indica que  $k = \lceil \log i \rceil$ . Per tant, el cost del bucle és  $\Theta(\log i)$ . Remarquem també que, si el bucle s'atura perquè  $b \geq n$ , el mateix raonament és vàlid.

Finalment, vegem el cost de la crida a *bin\_search*. En el cas pitjor  $b \geq n - 1$  i, per tant, el cost de la crida és  $\Theta(\log(b - b/2 + 1))$ . Com que després de  $k$  voltes,  $b$  val  $2^k$ , si el bucle ha fet  $k$  voltes el cost de la crida a *bin\_search* serà  $\Theta(\log(2^k - 2^{k-1} + 1)) = \Theta(\log(2^{k-1} + 1))$ . Sabem que el nombre de voltes és  $k = \lceil \log i \rceil$ , i per tant el cost de la crida a *bin\_search* és  $\Theta(\log i)$ .

Resumint, tot plegat té un cost en cas pitjor de  $\Theta(\log i)$ .