

**Cognoms**

**Nom**

**DNI**

**Examen Final AP3**

**Duració: 3 hores**

**07/01/2020**

- 
- *L'enunciat té 5 fulls, 10 cares i 4 problemes.*
  - *Poseu el vostre nom complet i número de DNI a cada full.*
  - *Contesteu tots els problemes en el propi full de l'enunciat i a l'espai reservat.*
  - *A no ser que es digui el contrari, cal justificar les respostes.*
- 

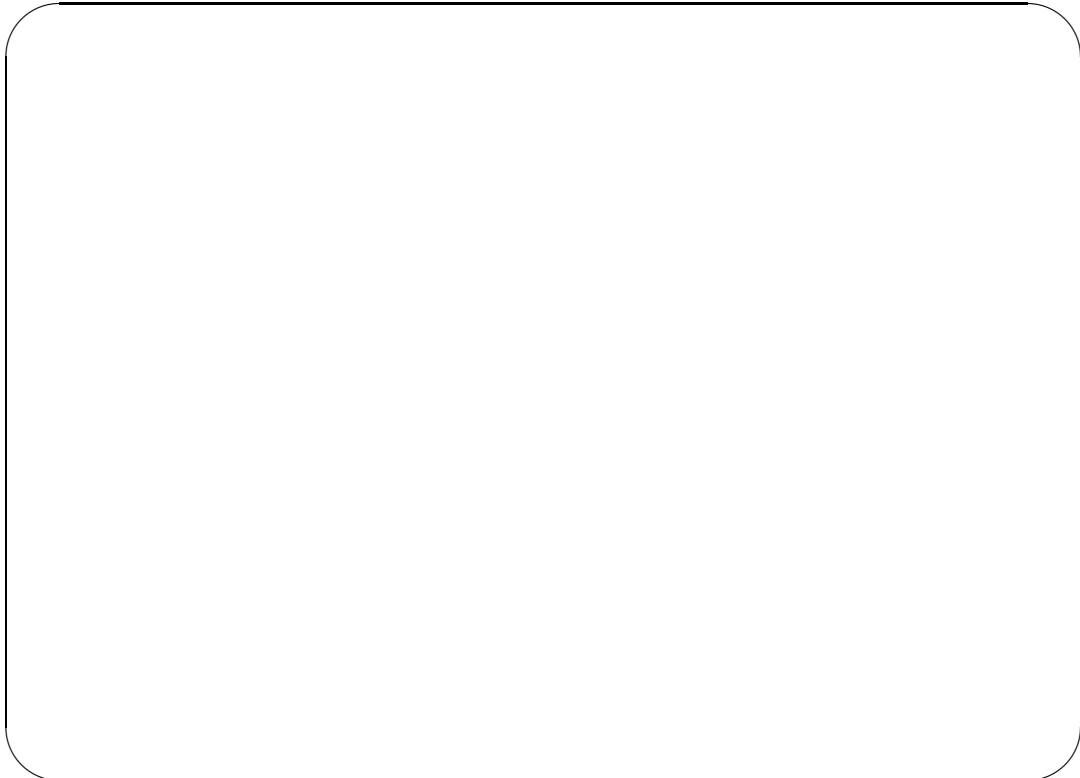
**Problema 1**

**(2 punts)**

- (a) (1 pt.) Doneu un DFA el llenguatge del qual siguin les seqüències binàries que, interpretades com a nombres naturals, són múltiples de 3. Per exemple, les paraules 0, 11, 00110 pertanyen al llenguatge, donat que representen els nombres 0, 3 i 6 respectivament. La paraula buida  $\lambda$  també forma part del llenguatge, ja que representa el nombre 0. En canvi, les paraules 1, 010, 101 **no** pertanyen al llenguatge, donat que representen els nombres 1, 2 i 5 respectivament.

Podeu descriure el DFA amb una taula de transicions o amb un diagrama de transicions. En qualsevol cas, indiqueu clarament l'estat inicial i els estats d'acceptació. Podeu usar els convenis de notació vistos a classe.

No cal justificar la resposta.



- (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 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.

Recordeu:  $k$ -SAT és el problema de, donada una fórmula proposicional en CNF amb com a molt  $k$  literals per clàusula, decidir si és satisfactible o no.

- (1) Tot problema de la classe P es pot decidir amb un algorisme de cost polinòmic.
- (2) Hi ha problemes de la classe NP que es poden decidir amb un algorisme de cost polinòmic.
- (3) Tot problema NP-complet es pot decidir amb un algorisme de cost exponencial.
- (4) Hi ha problemes que es poden reduir a 2-SAT i també a 3-SAT.
- (5) Hi ha problemes que es poden reduir a 3-SAT però no a 4-SAT.

	(1)	(2)	(3)	(4)	(5)
CERT					
FALS					

**Cognoms**

**Nom**

**DNI**

**Problema 2**

**(2 punts)**

Donat un vector  $a[0..n-1]$  de  $n > 0$  enters, el modifiquem de la manera següent: escollim un índex  $i$  amb  $0 \leq i < n$ , i reemplacem  $a[i]$  per  $-a[i]$ . Volem calcular la suma més gran dels elements del vector que es pot aconseguir després d'aplicar aquesta operació  $k > 0$  vegades.

Per exemple, si  $a = [3, -1, 0, 2]$  i  $k = 3$ , llavors la resposta és 6. Aquest valor es pot aconseguir escollint els índexos 1, 2, 2, la qual cosa dóna lloc al vector  $[3, 1, 0, 2]$ , que té suma 6.

- (a) (1 pt.) Si  $k = 1$ , quin índex podem escollir per obtenir la suma màxima? Demostreu-ho.

- (b) (1 pt.) Ompliu el buit de la funció següent per resoldre el problema. Es valorarà l'eficiència.

```
int max_sum(const vector<int>& a, int k) {
```

```
}
```

**Cognoms****Nom****DNI****Problema 3****(4 punts)**

En el problema de la motxilla disposem d'una motxilla amb capacitat  $W > 0$  i de  $n$  objectes, cadascun amb un pes  $w_k > 0$  i un valor  $v_k > 0$  (on  $0 \leq k < n$ ). Es tracta de trobar el valor acumulat màxim que es pot aconseguir amb un subconjunt d'objectes el pes total del qual no excedeixi la capacitat de la motxilla. És a dir, es tracta de trobar el màxim  $\sum_{k \in S} v_k$  on  $S \subseteq \{0, \dots, n-1\}$  és tal que  $\sum_{k \in S} w_k \leq W$ .

Suposem que per tot  $0 \leq k < n$  es compleix que  $w_k \leq W$  (doncs altrament l'objecte  $k$ -èsim mai podrà formar part de la selecció òptima).

- (a) (1 pt.) Ompliu els buits del programa següent per a resoldre el problema de la motxilla.

```

int n, W;
vector<int> v, w;

int opt1() {
    vector<vector<int>> m(n+1, vector<int>(W+1,  ));
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j <  ; ++j)
            m[i][j] = m[i-1][j];
        for (int j =  ; j <= W; ++j)
            m[i][j] = max(  , m[i-1][j]);
    }
    return  ;
}

int main() {
    cin >> n >> W;
    v = w = vector<int>(n);
    for (int& x : v) cin >> x;
    for (int& x : w) cin >> x;
    cout << opt1() << endl;
}

```

- (b) (0.5 pts.) Raoneu quin és el cost asimptòtic en temps d'*opt1* en funció de  $n$  i  $W$ .

- (c) (0.6 pts.) Donats  $i$  tal que  $0 \leq i \leq n$  i un valor  $V$  tal que  $V \leq \sum_{k=0}^{i-1} v_k$ , definim  $c(i, V)$  com el  $C$  més petit tal que hi ha un subconjunt dels  $i$  primers objectes amb pes total  $C$  i valor com a mínim  $V$ , o sigui, un subconjunt  $S \subseteq \{0, \dots, i-1\}$  tal que  $\sum_{k \in S} w_k = C$  i  $\sum_{k \in S} v_k \geq V$ . Completeu la recurrència següent.

$$c(i, V) = \begin{cases} \boxed{\phantom{00000000000000000000}} & \text{si } V \leq 0 \\ \boxed{\phantom{00000000000000000000}} & \text{si } V > \sum_{k=0}^{i-2} v_k \\ \boxed{\phantom{00000000000000000000}} & \text{altrament} \end{cases}$$

- (d) (0.9 pts.) Ompliu els buits del programa següent per a resoldre el problema de la motxilla, on `int c(int i, int V)` implementa la funció de l'apartat anterior.

```
int n, W;
vector<int> v, w, s;
vector<vector<int>> cc;
```

```
int c(int i, int V) {
```

```
}
```

```
int opt2() {
    s = vector<int>(n+1, 0);
    for (int k = 1; k <= n; ++k) s[k] = s[k-1] + v[k-1];
    int S = s.back();
    cc = vector<vector<int>>(n+1, vector<int>(S+1, -1));
    int V = 0;
    while (  ) ++V;
    return  ;
}
```

```
int main() {
    cin >> n >> W;
    v = w = vector<int>(n);
    for (int& x : v) cin >> x;
    for (int& x : w) cin >> x;
    cout << opt2() << endl;
}
```

**Cognoms**

**Nom**

**DNI**

- (e) (0.5 pts.) Raoneu quin és el cost asimptòtic en temps d' $opt2$  en funció de  $n$ , de  $W$  i de  $S = \sum_{k=0}^{n-1} v_k$  en el cas pitjor.

- (f) (0.5 pts.) Si volem resoldre una instància del problema de la motxilla i  $W$  és un valor petit i  $S$  és un valor gran, quin dels dos programes anteriors escolliríeu? I si  $W$  és un valor gran i  $S$  és un valor petit? Raoneu la resposta.

*Aquesta cara estaria en blanc intencionadament si no fos per aquesta nota.*



Cognoms

Nom

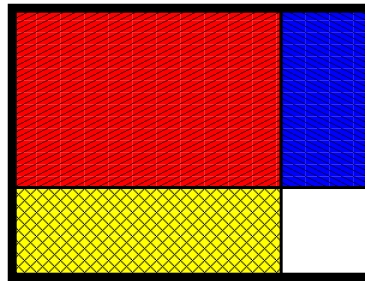
DNI

#### Problema 4

(2 punts)

Avui és l'endemà del dia de Reis, i toca tornar a posar ordre a la casa. Finalment els Reis Mags no han portat carbó, sinó una pila de joguines que ara cal guardar. Per fer-ho, disposem d'un calaix de dimensions  $n \times m$ , on  $n, m > 0$ . Tenim  $p$  joguines (amb  $p > 0$ ), cadascuna de les quals ve en una caixa rectangular de dimensions  $a_k \times b_k$ , on per cada  $0 \leq k < p$  es compleix  $0 < a_k \leq n$  i  $0 < b_k \leq m$ . Per simplificar, ignorarem l'alçada del calaix i de les caixes, i només considerarem dues dimensions, l'amplada i la fondària. També per simplificar assumirem que les caixes no poden ser rotades (és a dir, una caixa de dimensions  $1 \times 2$  **no** és igual que una caixa de dimensions  $2 \times 1$ ).

Per exemple, si  $n = 3$ ,  $m = 4$ ,  $p = 3$  i les caixes de joguines tenen dimensions  $2 \times 3$ ,  $2 \times 1$  i  $1 \times 3$  respectivament, aleshores una possible manera de col·locar les caixes al calaix (mirant el calaix des de dalt) és aquesta:



Observeu que el calaix no té per què quedar ple del tot.

Ompliu els buits del programa següent per a resoldre el problema de col·locar les caixes de joguines al calaix.

```
int n, m, p;
vector<int> a, b, u;
vector<vector<int>> s;

bool can_place_at(int k, int i, int j) {
    if (  ) return false;
    for (int r = i; r < i + a[k]; ++r)
        for (int c = j; c < j + b[k]; ++c)
            if (  ) return false;
    return true;
}

void fill(int k, int i, int j, int v) {
    for (int r = i; r < i + a[k]; ++r)
        for (int c = j; c < j + b[k]; ++c)
            s[r][c] = v;
}
```

```

bool bt(int i, int j, int left) {
    if (  ) return false;
    if (i == n) {
        for (int r = 0; r < n; ++r) {
            for (int c = 0; c < m; ++c)
                cout << ' ' << s[r][c];
            cout << endl;
        }
        return true;
    }
    if (j == m) return  ;
    if (s[i][j] != -1) return  ;
    for (int k = 0; k < p; ++k) {
        if (not u[k] and can_place_at(k, i, j)) {
            u[k] = true;
            fill(k, i, j, k);
            if (bt(i, j+1,  )) return true;
            fill(k, i, j, -1);
            u[k] = false;
        }
    }
    return  ;
}

int main() {
    cin >> n >> m >> p;
    a = b = vector<int>(p);
    int left = 0;
    for (int k = 0; k < p; ++k) {
        cin >> a[k] >> b[k];
        left += a[k] * b[k];
    }
    s = vector<vector<int>>(n, vector<int>(m,  ));
    u = vector<int>(p, false);
    if (not bt(, , left)) cout << "No solution" << endl;
}

```