# Solucions de la Final

Olimpíada Informàtica Catalana 2018

# Problema 1: Popurri

## Bits

*Quants valors diferents es poden representar amb un bit?*

Només cal saber que un bit pot valer 0 ó 1, per tant només es poden codificar dos valors diferents.

## Nombres en binari

*Associeu cada nombre en binari al seu equivalent en decimal.*

Sigui $a_n a_{n-1} \dots a_1 a_0$ un nombre en binari, on els $a_i$ són cadascun dels seus dígits. El nombre equivalent en decimal el podem obtenir fent $\sum a_i 2^i$, però en aquest cas no cal fer la conversió: podem ordenar totes dues columnes i aquesta serà la resposta correcta.

## Acrònim

*Què vol dir l'acrònim CPU en el món de la informàtica?*

CPU vol dir *Central Processing Unit*, que en català seria *Unitat Central de Procés*.

## Història

*Ordeneu de més antic a més modern els dispositius de còmput següents: Computador ENIAC, Calculadora de Blaise Pascal (Pascalina), Màquina Analítica de Charles Babbage, Primer Macintosh, Àbac*

| Àbac | Difícil de datar, però segles abans de Crist |
|---|---:|
| Pascalina | 1642 |
| Màquina analítica | 1837 |
| ENIAC | 1945 |
| Macintosh | 1984 |

## Gent

*Associeu cada invent/producte amb el seu creador.*

| Màquina de Turing | Alan Turing |
|---|---:|
| Apple | Steve Jobs |
| Windows | Bill Gates |
| Facebook | Mark Zuckerberg |
| WiFi, GPS, Bluetooth... | Hedy Lamarr |

# Problema 2: Mitjanes Polilingüístiques

## Mitjana en C++

*Completeu aquest codi en C++ amb el mínim nombre de caràcters perquè retorni la mitjana dels elements d'un vector no buit d'enters* `v`.

```
double mitjana(const vector<int>& v) {
    double s = 0;
    int n = v.size();
    for (int i = 0; i < n; ++i) {
        s += v[i];
    }
    return s / n;
}
```

## Mitjana en C

*Completeu aquest codi en C amb el mínim nombre de caràcters perquè retorni la mitjana dels* `n` *elements d'un array d'enters* `v`*, amb* $\texttt{n} > 0$.

```
double mitjana(int v[]) {
    double s = 0;
    for (int i = 0; i < n; ++i) {
        s += v[i];
    }
    return s / n;
}
```

## Mitjana en Python

*Completeu aquest codi en Python3 amb el mínim nombre de caràcters perquè retorni la mitjana dels elements d'una llista no buida d'enters* `L`.

```
def mitjana(L):
    s = 0
    n = len(L)
    for x in L:
        s += x
    return s / n
```

## Mitjana en Java

*Completeu aquest codi en Java amb el mínim nombre de caràcters perquè retorni la mitjana dels elements d'un vector no buit d'enters* `v`.

```
static double mitjana(int[] v) {
    double s = 0;
    int n = v.size();
    for (int i = 0; i < n; ++i) {
        s += v[i];
    }
    return s / n;
}
```

# Problema 3: Àngels i dimonis

*En un temple hi ha un grup de personatges, cadascun dels quals pot ser un àngel o un dimoni. Els àngels sempre diuen la veritat, i els dimonis sempre menteixen. Per exemple, si X diu "tant Y com Z són àngels", i X és un dimoni, llavors almenys un entre Y i Z ha de ser un dimoni.*

*Hi ha cinc personatges, identificats amb les lletres* `A`*,* `B`*,* `C`*,* `D` *i* `E`*. Cadascun diu el següent:*

`A` *"Tant* `B` *com* `C` *són dimonis."*

`B` *"*`E` *és un dimoni."*

`C` *"*`A` *és un àngel."*

`D` *"Jo sóc un àngel."*

`E` *"Almenys un entre* `C` *i* `D` *és un àngel."*

Si `C` fos un angel, `A` també ho seria i arribem a una contradicció pel que diu `A`. Necessàriament tenim que `C` és un dimoni i per tant `A` també ho és. Ara sabem pel que diu `A` que o bé `B` o bé `C` (o tots dos) són àngels, però com sabem que `C` no ho és arribem a la conclusió que `B` és un àngel. De `B` sabem que `E` és un dimoni i d'aquí obtenim que `D` també ho és.

Observeu que el que diu `D` no aporta cap informació.

# Problema 4: Zeros i uns

L'únic nombre que es coneix que compleixi la propietat és el 82000. La solució esperada consisteix en fer un programa que iteri sobre els nombres fins a trobar un que compleixi les restriccions, com es fa a `pb4.cpp`.

# Problema 5: Primers de Wilson

Saber el nombre de primers de Wilson que existeixen és un problema obert en matemàtiques. Es conjectura que hi ha infinits, però els únics coneguts són 5, 13 i 563. Així doncs, la solució esperada consisteix en iterar sobre els nombres fins a 10000 i mirar si són primers de Wilson. Com que els factorials que apareixen són nombres molt grans, és força més senzill fer el codi en Python. Mireu una possible implementació a `pb5.py`.

# Problema 6: Triangle rectangle isòsceles

Aquest és un problema gràfic bastant bàsic, podeu trobar la solució oficial a `pb6.py`.

# Problema 7: Punts amb coordenades múltiples

Només cal iterar per tots els pixels de la imatge i comprovar si $x$ és múltiple de $y$ o a l'inrevés. Cal anar amb compte de no fer divisions per zero i per tant cal tractar els casos on un dels dos nombres és zero de forma especial, mireu `pb7.py` per veure com fer-ho.

## Problema 8: Fractal de cercles

Tot i que els fractals normalment es programen més fàcilment de forma recursiva, en aquest cas és potser més senzill de forma iterativa, ja que el fractal només avança per una branca. La principal dificultat del problema es troba en calcular bé les coordenades on anirà cada cercle. Podeu veure la solució proposada a `pb8.py` per comprovar les coordenades.

## Problema 9: Fora de joc?

El primer que cal veure d'aquest problema és que les regles del fora de joc no són les reals, sinó una versió simplificada. Tot i que és un problema un tant pesat de programar pels molts detallets que té, no és realment complicat. Cal anar amb compte a l'hora de pintar els diferents elements de seguir l'ordre adequat (els jugadors es pinten després de les línies del camp, per exemple). Podeu trobar a `pb9.py` una solució correcta.

## Problema 10: Mòria

Aquest problema és un recorregut en una graella i per tant es pot resoldre indistintament amb una cerca en amplada o en profunditat. La solució presentada a `pb10.py` fa una cerca en profunditat mentre marca a una matriu quines caselles han estat visitades per pintar-les després. Cal recordar pintar les voreres del mapa sense importar si són conegudes o no.

## Problema 11: Lletra del DNI

Problema molt senzill on només cal fer el que s'indica a l'enunciat. Podeu trobar solucions a `pb11.cpp` i a `pb11.py`.

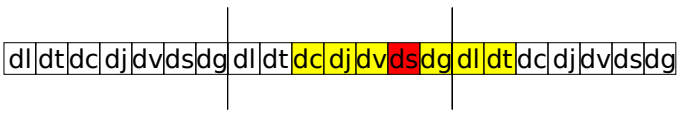
## Problema 12: Tres dígits

Hi ha moltes maneres de resoldre aquest problema. Una possible solució és mirar primer el dígit més petit que no sigui zero, escriure'l i després escriure la resta de dígits en ordre, com es fa a `pb12.py`.

Una altra opció és provar totes les permutacions dels dígits en ordre i imprimir la primera que no comenci per zero, com podeu trobar a `pb12.cpp`.

## Problema 13: Agenda perduda

Si fixem un dia de la setmana per una reunió, el conjunt de dies possibles per l'altra reunió conté una única vegada tots els altres dies de la setmana, com es veu a la següent imatge:



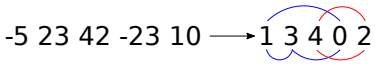

Per tant només cal comprovar les dues opcions possibles pels dies donats i imprimir la que tingui sentit. Teniu solucions en Python3 i C++ a `pb13.py` i a `pb13.cpp`.

# Problema 14: Predictor de patologies

La dificultat del problema és entendre el que cal fer, tot i que amb la xerrada del matí hauria de ser molt senzill d'entendre. El problema és autocontingut en qualsevol cas i només cal comptar quants cops apareixen dos caràcters a una columna de la matriu per cada pregunta i després imprimir un resultat en funció de la fórmula de l'enunciat. A `pb14.py` i `pb14.cpp` podeu trobar solucions.

# Problema 15: Intercanvis (2)

Aquesta és una versió més complicada del problema `Intercanvis` que va aparèixer anteriorment a l'OIcat 2018. Com que tots els nombres són diferents, podem convertir aquest problema a l'anterior assignant els nombres en ordre a una permutació normal i després resolent com es va explicar per `Intercanvis`, com es veu a la imatge i podeu trobar a `pb15.cpp` i a `pb15.py`.




# Problema 16: Festa salvatge (2)

Aquesta és una versió més complicada del problema `Festa salvatge` que va aparèixer anteriorment a l'OIcat 2018. La idea conceptual és que una planta pot substituir a la primera si un cop ordenades les lletres, la candidata és menor o igual a cada posició. Aquesta solució és massa lenta ja que si el codi intenta ordenar serà massa lent. La manera eficient de fer-ho és fer servir que només hi ha 26 lletres diferents i evitar ordenar, com es fa a `pb16.cpp`.

# Problema 17: Cercle interior?

Primer cal veure que l'únic cercle candidat a ser interior és aquell que tingui radi mínim. Si hi ha més d'un amb radi mínim, qualsevol servirà ja que per haver-ne un d'interior, caldrà que siguin el mateix cercle.

Un cop tenim un candidat, només cal veure si és interior a tots els altres. Això es pot veure fàcilment:

$$(x_1, y_1, r_1) \subseteq (x_2, y_2, r_2) \Leftrightarrow r_1 \leq r_2,\ (x_1 - x_2)^2 + (y_1 - y_2)^2 \leq (r_2 - r_1)^2$$

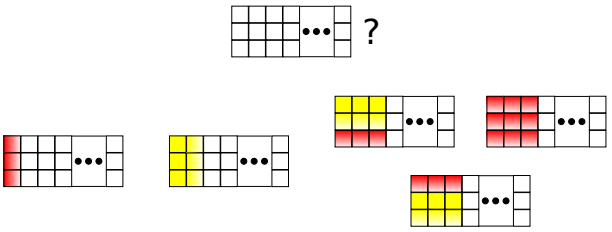
A `pb17.cpp` podeu trobar un codi que fa això.

# Problema 18: ABBA (2)

Aquesta és una versió més complicada del problema `ABBA` que va aparèixer anteriorment a l'OIcat 2018. Igual que al problema anterior, cal programar un backtracking per trobar totes les paraules, però com que aquí ens demanen les que no tinguin una certa subparaula, podem anar podant les branques de la recursió en el moment en que continguin les seqüències prohibides, com fem a `pb18.cpp`. Sense això el codi és massa lent i no entra per temps.

# Problema 19: Totxanes (2)

Aquesta és una versió més complicada del problema `Totxanes` que va aparèixer anteriorment a l'OIcat 2018, ja que ara disposem d'un nou tipus de peça.



Seguint el mateix raonament que al problema anterior, a la imatge veiem les diferents maneres de començar a posar una totxana, de forma que obtenim la recurrència

$$F(n) = F(n-1) + F(n-2) + 3F(n-3)$$

que caldrà programar amb programació dinàmica fent servir que $F(0) = F(1) = 1$ i $F(2) = 2$. Podeu veure un exemple amb memoization a `pb19.cpp` i un de forma iterativa a `pb19.py`.

# Problema 20: Globglogabgalab

Per $n = 2$ la resposta és clarament 2 i es pot demostrar que la resposta és $2(n-2)$ per $n \geq 3$. La demostració de que no es pot fer amb menys s'omet per ser bastant pesada. Vegem doncs que es pot fer amb aquest nombre de preguntes.

Considerem les següents preguntes: $2, 3, \ldots, n-2, n-1, n-1, n-2, \ldots, 3, 2$. Clarament hi ha $2(n-2)$, veiem que són solució. Representarem gràficament les posicions on el monstre pot ser. Inicialment tenim això:

| x | x | x | x | x | ... | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|

Preguntant per 2, ens queda això:

| | x | x | x | x | ... | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|

Preguntant per 3:

| x | | x | x | x | ... | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|

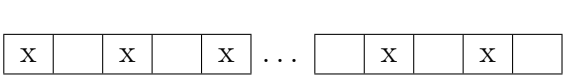
Ara toca el 4:

| | x | | x | x | ... | x | x | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|

A aquestes alçades es pot veure doncs que cada pregunta canvia la paritat dels forats blancs. Més concretament, després de preguntar per $i$, tenim forats a tots els quadrats menors que $i$ que tenen diferent paritat. Seguim fins a $n-2$:

| | x | | x | | ... | x | | x | x | x |
|---|---|---|---|---|---|---|---|---|---|---|

És important explicitar en aquest punt que el primer bloc depèn de la paritat de $n$ i que hem triat una de les dues opcions arbitràriament perquè no afecta a la demostració. Preguntem ara per $n-1$:




Es veu clar per tot el que hem vist fins ara que després de preguntar per $n-1$, l'afirmació anterior continua sent certa però necessàriament tindrem un forat a la posició $n$. A partir d'aquest moment la paritat del monstre està fixada i anirà alternant amb cada pregunta. És per això que si tornem a fer les preguntes $n-1, n-2, \ldots, 3, 2$, sempre acabarem trobant al monstre. El codi no té massa misteri, podeu veure una solució a `pb20.cpp`