Exercicis d'Informàtica

Jordi Cortadella

16 d'octubre de 2018

Aquesta és una col·lecció d'exercicis que cobreixen una part important del temari del curs d'Informàtica i complementen les col·leccions de problemes proposats en el `jutge` i realitzats a les classes de laboratori.

Els exercicis tracten principalment d'aquells aspectes més teòrics (recursivitat, invariants, complexitat algorísmica, etc) que no poden ser fàcilment coberts amb exercicis de laboratori. Estan agrupats per temes i l'ordre d'aparició no suposa necessàriament l'ordre recomenat per a la seva resolució. Es suggereix que l'estudiant faci exercicis de diversos temes tant aviat hagi adquirit els coneixements necessaris per a resoldre'ls.

Recursivitat

1. Dissenyeu la funció

   ```
   bool multiple_11(int n);
   ```

   on $n \geq 0$. La funció ha de retornar *cert* si n és múltiple d'11 i *fals* en cas contrari. Per dissenyar l'algorisme cal fer servir aquesta propietat *recursiva*:

   > Un número és múltiple de 11 si la diferència entre la suma dels dígits a les posicions parells i la suma dels dígits a les posicions senars és múltiple de 11.

   Per exemple, 91527381 és múltiple de 11 donat que (9+5+7+8) - (1+2+3+1) = 22. En canvi 835061 no és múltiple de 11 donat que (8+5+6)-(3+0+1) = 15.
   Dissenyeu la funció sense fer servir cap funció auxiliar ni cap operació de divisió o mòdul per 11 (és poden fer operacions de divisió i mòdul per 10). Es valorarà l'ús de la recursivitat, tenint en compte que cal respectar l'especificació de la funció ($n \geq 0$) en fer la crida recursiva.

   Solució:

   ```
   bool multiple_11(int n) {
       if (n < 11) return n == 0;
       int suma = 0;
       int signe = 1;
       while (n > 0) {
           suma = suma + signe * n%10;
           n = n/10;
           signe = -signe;
       }
       if (suma < 0) suma = -suma;
       return multiple_11(suma);
   }
   ```

2. Dissenyeu la funció

```
int combinacions(int n, int k);
```

on $n \geq k \geq 0$. La funció ha de calcular

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Dissenyeu la funció recursivament, sense fer servir cap funció auxiliar ni cap instrucció `for` o `while`.

Solució: Es poden proposar diferents solucions basades en diverses definicions recursives:

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1} = \frac{n-k+1}{k}\binom{n}{k-1} = \frac{n}{n-k}\binom{n-1}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Per cadascuna cal identificar el cas bàsic i el cas recursiu. Convé sempre fer les divisions al final per evitar trucaments de expressions enteres. Aquesta és una de les possibles solucions:

```
int combinacions(int n, int k) {
    if (k == 0) return 1;
    return combinacions(n - 1, k - 1)*n/k;
}
```

3. La funció següent retorna el nombre de factors que té la descomposició en factors primers del nombre n, on $n \geq 1$.

```
int numFactors(int n) {
  return nf(n, 2);
}
```

Exemples:

`numFactors(12) = 3`,	($12 = 2^2 \times 3$)
`numFactors(150) = 4`,	($150 = 2 \times 3 \times 5^2$)
`numFactors(17) = 1`,	(17 és primer)
`numFactors(1) = 0`,	(1 no es pot descompondre en factors primers)
`numFactors(2016) = 8`,	($2016 = 2^5 \times 3^2 \times 7$)

Dissenyeu la funció auxiliar `nf` per tal que `numFactors` compleixi la seva especificació. Cal que `nf` sigui recursiva i no cridi a cap altra funció. No es permet la utilització de cap tipus d'instrucció iterativa. **Pista:** El segon paràmetre de `nf` representa *el següent divisor a explorar*.

Solució:

```
int nf(int n, int d) {
  // d es el divisor mes petit que queda per explorar
  if (n == 1) return 0;
  if (d*d > n) return 1; // es primer, no cal explorar mes
  if (n%d == 0) return 1 + nf(n/d, d);
  return nf(n, d + 1);
}
```

Nota: El segon cas (`d*d > n`) fa que la funció sigui eficient.

4. La funció de Fibonacci $F(n)$ es defineix per a $n \geq 0$ de la manera següent:

$$F(n) = \begin{cases} n & \text{si } n \leq 1 \\ F(n-1) + F(n-2) & \text{si } n > 1 \end{cases}$$

Aquesta definició dóna lloc a la sèrie:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

Proposeu un disseny recursiu *eficient* de la funció de Fibonacci que sigui compatible amb la crida següent:

```
int Fibonacci(int n) {
  return Fib_rec(n, 1, 0);
}
```

Dissenyeu la funció auxiliar `Fib_rec` sense fer servir cap instrucció iterativa. La funció `Fib_rec` només podrà cridar-se a sí mateixa. Es valorarà la correctesa, la senzillesa i l'eficiència.

Solució:

```
// fi1 i fi contenen fib(i+1) i fib(i)
// n repesenta el nombre de termes que manquen per sumar
int Fib_rec(int n, int fi1, int fi) {
  if (n == 0) return fi;
  return Fib_rec(n - 1, fi1 + fi, fi1);
}
```

5. La següent funció recursiva calcula el màxim comú divisor de dos nombres. Utilitza una variant de l'algorisme d'Euclides on només es fan servir multiplicacions i divisions per 2, a més de sumes i restes. Completar els casos que manquen:

```
int mcd(int a, int b) {
  if (a == 0 or b == 0) return ..............................;

  if (a%2 == 0 and b%2 == 0) return .........................;

  if (a%2 == 0) return.......................................;

  if (b%2 == 0) return ......................................;
  if (a < b) return mcd(a, (b - a)/2);
  return mcd(b, (a - b)/2);
}
```

Solució:

```
int mcd(int a, int b) {
    if (a == 0 or b == 0) return a + b;
    if (a%2 == 0 and b%2 == 0) return 2*mcd(a/2,b/2);
    if (a%2 == 0) return mcd(a/2, b);
    if (b%2 == 0) return mcd(a,b/2);
    if (a < b) return mcd(a, (b - a)/2);
    return mcd(b, (a - b)/2);
}
```

6. Dissenyeu la funció

```
int multinomial(int a, int b, int c)
```

on $a \geq b \geq c \geq 0$. La funció calcula el coeficient multinomial definit de la manera següent:

$$\texttt{multinomial(a, b, c)} = \binom{a+b+c}{a,b,c} = \frac{(a+b+c)!}{a!\ b!\ c!}$$

Dissenyeu la funció recursivament, sense fer servir cap instrucció `for` o `while`. En fer la crida recursiva, cal respectar la precondició $a \geq b \geq c \geq 0$. Cal que la funció sigui eficient, evitant calcular tots els factorials.

Solució: Es pot fer servir la igualtat següent, per $c > 0$:

$$\begin{aligned} \texttt{multinomial(a, b, c)} &= \frac{(a+b+c)!}{a!\ b!\ c!} = \frac{a+b+c}{c} \cdot \frac{(a+b+c-1)!}{a!\ b!\ (c-1)!} = \\ &= \frac{a+b+c}{c} \cdot \texttt{multinomial(a, b, c - 1)} \end{aligned}$$

```
int multinomial(int a, int b, int c) {
    if (b == 0) return 1;    // Sabem que c == 0
    if (c == 0) return (a + b)*multinomial(a, b - 1, 0)/b;
    return (a + b + c)*multinomial(a, b, c - 1)/c;
}
```

Una solució alternativa seria la de utilitzar coeficients binomials amb la igualtat següent:

$$\binom{a+b+c}{a,b,c} = \binom{a+b+c}{c} \cdot \binom{a+b}{b}$$

```
int binomial(int n, int k) {
  if (k == 0) return 1;
  return n*binomial(n - 1, k - 1)/k;
}

int multinomial(int a, int b, int c) {
    return binomial(a + b + c, c)*binomial(a + b, b);
}
```

7. Considereu una funció amb l'especificació següent:

```
// n > 0
// Retorna la longitut de la cadena mes llarga de zeros
// continguda en n representat en base 10
int MaxZeros(int n);
```

Exemples:
`MaxZeros`(300800050)=3, `MaxZeros`(12)=0, `MaxZeros`(60090000)=4

Dissenyeu una solució iterativa. Es valorarà la senzillesa de l'algorisme.
Podeu utilitzar la funció `max` per calcular el màxim de dos nombres.

Solució:

```
int MaxZeros(int n) {
  int maxz = 0;
  int actual = 0;
  // Invariant:
  //   maxz es la cadena mes llarga de zeros trobada fins ara
  //   actual es la longitut de la cadena de zeros actual
  while (n > 0) {
    if (n%10 == 0) {
      ++actual;
      maxz = max(actual, maxz);
    } else actual = 0;
    n = n/10;
  }
  return maxz;
}
```

Dissenyeu una versió recursiva utilitzant una funció auxiliar amb paràmetres addicionals i sense utilitzar cap instrucció iterativa.

Solució:

```
int MaxZerosRec(int n, int maxz, int actual) {
  if (n == 0) return maxz;
  if (n%10 != 0) return maxzeros(n/10, maxz, 0);
  return maxzeros(n/10, max(maxz, nz + 1), nz + 1);
}

int MaxZeros(int n) {
  return MaxZerosRec(n, 0, 0);
}
```

Una solució alternativa, només amb un paràmetre addicional a la funció recursiva, podria ser la següent:

Solució:

```
int MaxZerosRec(int n, int maxz) {
  if (n == 0) return maxz;
  if (n%10 != 0) return max(maxz, maxzeros(n/10, 0));
  return maxzeros(n/10, maxz + 1);
}

int MaxZeros(int n) {
  return MaxZerosRec(n, 0);
}
```

8. Volem dissenyar una funció que transformi la representació d'un nombre de la manera següent:

> *Els dígits amb valor parell que ocupin posicions parelles i el dígits amb valor senar que ocupin posicions senars seran substituïts per zeros. La resta de dígits es mantindrà igual. Es considera que la posició de les unitats és la 0, la de les desenes és la 1, la de les centenes és la 2, etc.*

Exemples:

$$\begin{aligned} 12345678 &\rightarrow 0 \\ 87654321 &\rightarrow 87654321 \\ 11223344 &\rightarrow 1200340 \\ 653867112 &\rightarrow 3800100 \end{aligned}$$

La funció inicial és la següent:

```
int Transform(int n) {
  return T(n, ...);
}
```

on `T` és una funció recursiva que té dos paràmetres. Dissenyeu la funció `T` amb les pautes següents:

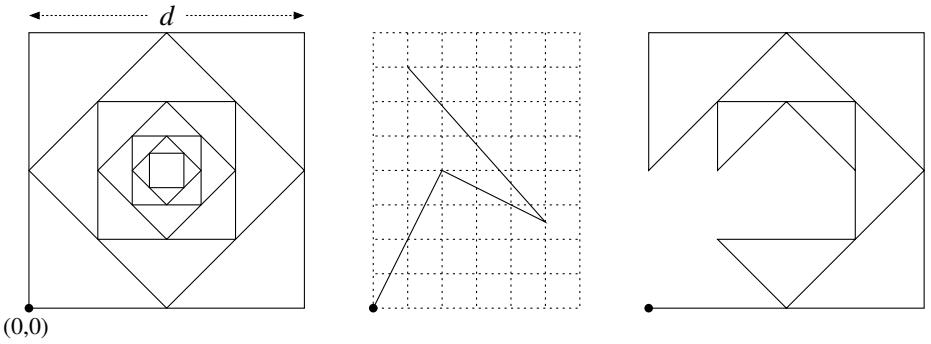
- `T` ha de tenir exactament dos paràmetres. Expliqueu el significat del segon paràmetre.
- `T` ha de ser purament recursiva. No pot fer servir cap instrucció iterativa.

Es valorarà la correctesa, senzillesa i elegància de la solució.

Solució:

```
int T(int n, int i) {
  // i representa la posicio del darrer digit
  // (inicialment 0 en la primera crida)
  if (n == 0) return 0;
  int x = T(n/10, i+1)*10;
  if (n%2 != i%2) x += n%10;
  return x;
}
```

9. **Quadrat dins de quadrat dins de quadrat ...**



Volem dissenyar un algorisme que dibuixi un conjunt de quadrats imbricats tal com mostra la figura de l'esquerra. Els costats del quadrat més gran han de tenir llargada d i el vèrtex inferior esquerra ha d'estar a l'origen (0,0).
Disposem d'una llibreria gràfica que permet moure un bolígraf sobre el paper. La llibreria només té dues funcions:

- `home()`: Posa el bolígraf a l'origen (sense dibuixar).
- `draw(`$\Delta x, \Delta y$`)`: Dibuixa movent el bolígraf des de la posició actual (x, y) fins la posició $(x + \Delta x, y + \Delta y)$ en línia recta.

Per exemple, la figura del mig representa el dibuix fet amb la seqüència següent:

```
home(); draw(2,4); draw(3,-1.5); draw(-4,4.5);
```

Restriccions:

- Volem minimitzar la despesa de tinta. Per tant, no volem dibuixar dues vegades la mateixa línia.
- L'algorisme ha de començar a dibuixar des de l'origen i ha d'acabar de dibuixar a l'origen.

A la dreta es mostra una possible trajectoria del bolígraf en un punt intermig d'execució de l'algorisme. Heu de fer que el vostre algorisme segueixi *exactament* aquest patró de dibuix.
Dissenyeu la funció amb l'especificació següent:

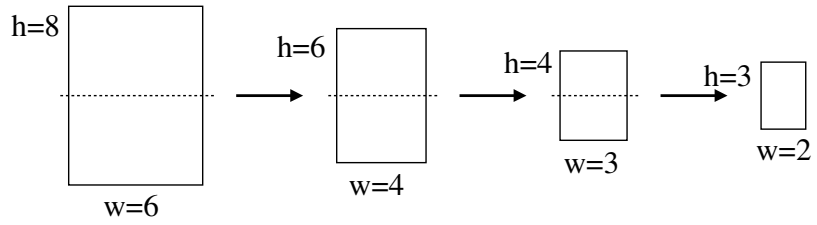
```
void drawSquares(int n, double d);
```

on $n \geq 0$ és el nombre de quadrats a dibuixar ($n = 7$ a l'exemple) i d és la longitut del costat del quadrat més gran.
Notes: Es recomana fer servir una funció auxiliar recursiva. Expliqueu clarament què és cada paràmetre de la funció. Es valorarà principalment la correctesa i la senzillesa de l'algorisme.

Solució:

```
void drawSquares(int n, double d) {
  home();
  drawRecursive(n, d, 0);
}

void drawRecursive(int n, double d, int i) {
// n i d tenen el mateix significat que a la funcio anterior.
// i es el numero de quadrat a dibuixar
// (0 el mes extern, n-1 el mes intern)

    if (i == n) return; // Cas basic: ja hem acabat

    if (i%2 == 0) { // Quadrat rectilini (sense rotar)
      draw(d, 0); draw(0,d); draw(-d,0); draw(0,-d/2);
      drawRecursive(n, d/2, i+1);
      draw(0, -d/2);
    } else { // Quadrat rotat 45 graus
      draw(d, d); draw(d,-d); draw(-d,-d); draw(-d/2,d/2);
      drawRecursive(n, d, i+1);
      draw(-d/2, d/2);
    }
}
```

10. Tenim un full de paper rectangular amb alçada h i amplada w. El paper té unes mides tals que $h \geq w \geq h/2$. Volem conèixer la mida del paper després de doblegar-lo n vegades reduint cada vegada la seva alçada a la meitat, tal com mostra l'exemple de la figura, en el que un paper s'ha doblegat 3 vegades.



Dissenyeu la funció següent per calcular la mida final del paper.

```
Pre:  h >= w >= h/2, n >= 0.
Post: h i w representen les mides del paper
      despres de ser doblegat n vegades
void doblegar(double& w, double& h, int n);
```

Important: El disseny ha de ser recursiu, sense fer servir cap funció auxiliar. Cal tenir en compte que els paràmetres w i h es passen per referència i que el resultat ha de ser rebut en aquests paràmetres.

Solució:

```
void  doblegar(double& w, double& h, int n) {
  if (n > 0) {
    int z = h;
    h = w;
    w = z/2;
    doblegar(w, h, n - 1);
  }
}
```

Esbrinar

11. Un estudiant ha entregat la següent implementació d'un procediment. Podeu explicar què fa? Quin cost té l'algorisme?

```
void que_fa_aixo(vector<int>& a, int x) {
    int i = 0;
    int j = a.size() - 1;
    while (i < j) {
        if (a[i] <= x) ++i;
        else {
            swap(a[i], a[j]);
            --j;
        }
    }
}
```

Solució: La funció reordena els elements del vector `a` de tal manera que els més petits o iguals que `x` queden a les posicions baixes i els més grans que `x` queden a les posicions altes.

$\leq x$	$> x$

La complexitat de l'algorisme és lineal en la mida del vector.

12. Considereu el programa següent:

```
// Pre: n > 0
// Retorna ...
int esbrina(int n) {
  while (n > 9) n = n/10;
  return n%10;
}

// Llegeix una sequencia de nombres positius (diferents de zero)
int main() {
  int n;
  cin >> n;
  int lsd = n%10;
  bool segueix = true;
  while (segueix and cin >> n) {
    segueix = esbrina(n) == lsd;
    lsd = n%10;
  }
  if (segueix) cout << "si" << endl;
  else cout << "no" << endl;
}
```

- Completar l'especificació de la funció `esbrina`:

  Solució: Retorna el dígit més significatiu de n representat en base 10.

- Es podria fer alguna simplificació del codi de la funció `esbrina`? Raonar la resposta.

  Solució: N'hi hauria prou amb la instrucció `return n`, donat que a la sortida del bucle n només té un dígit.

- Donar una seqüència d'entrada de 5 nombres diferents per la qual el programa escrigui `"si"`.

  Solució: 12345 54321 1 187 763

- Donar una seqüència d'entrada de 5 nombres diferents per la qual el programa escrigui `"no"`.

  Solució: 12345 5 532 4444 78

- Escriure una especificació per a la funció `main`:

  Solució: El programa llegeix una seqüència no buida de nombres positius. Escriu "si"en el cas que tots els nombres es puguin encadenar, és a dir, que el dígit de menys pes d'un nombre sigui igual al dígit de més pes del nombre següent. En cas que no tots els nombres es puguin encadenar, escriu "no".

---

13. Un estudiant ha entregat la següent implementació d'una funció. Podeu explicar què fa? Quin cost té l'algorisme? (suposeu que els vectors `a` i `b` no són buits i només contenen elements positius).

```
bool que_fa_aixo(const vector<int>& a, const vector<int>& b) {
    int diff = a[0] - b[0];
    int i = 1;
    int j = 1;
    while (i < a.size() and j < b.size() and diff != 0) {
        if (diff > 0) {
            diff = diff - b[j]; ++j;
        } else {
            diff = diff + a[i]; ++i;
        }
    }

    while (diff < 0 and i < a.size()) {
        diff = diff + a[i]; ++i;
    }

    while (diff > 0 and j < b.size()) {
        diff = diff - b[j]; ++j;
    }

    return diff == 0;
}
```

Solució: La funció retorna *cert* si existeixen dos prefixos dels vectors tals que la suma dels seus elements és la mateixa. Retorna *fals* en cas que aquests prefixos no existeixin.

14. Expliqueu què conté el vector `V` en relació al seu valor inicial després d'executar cadascuna de les funcions següents (Nota: no heu d'explicar com funciona l'algorisme internament):

```
// Pre: 0 <= i,j < V.size()
void r(vector<int>& V, int i, int j) {
  while (i < j) {
    swap(V[i], V[j]);
    ++i; --j;
  }
}
```

Solució: Gira les posicions $V[i \ldots j]$ del vector.

```
void QueFaAixo(vector<int>& V) {
  r(V, 0, V.size() - 2);
  r(V, 0, V.size() - 1);
}
```

Solució: Posa el darrer element del vector a la primera posició i desplaça les altres una posició més endavant.

```
// 0 <= k <= V.size()
void EncaraMes(vector<int>& V, int k) {
  r(V, 0, k - 1);
  r(V, k, V.size() - 1);
  r(V, 0, V.size() - 1);
}
```

Solució: Fa una rotació de tots els elements k posicions cap a l'esquerra (els elements que surten per l'esquerra tornen a entrar per la part dreta).

```
// 0 <= k <= V.size()
void LaQueMes(vector<int>& V, int k) {
  EncaraMes(V, k);
  EncaraMes(V, V.size() - k);
}
```

Solució: Aquesta funció no fa res (deixa el vector tal com estava al principi).

15. Suposant que $x \neq 0$ i $y \geq 0$, les dues funcions que es mostren a continuació fan el mateix.

```
int puzzle1(int x, int y) {
  if (y == 0) return 1;
  int z = puzzle1(x*x, y/2);
  if (y%2 == 1) z = z*x;
  return z;
}
```

```
int puzzle2(int x, int y) {
  int x2 = x*x, z = 1;
  while (y >= 2) {
    z = z*x2;
    y = y - 2;
  }
  if (y == 1) z = z*x;
  return z;
}
```

- Què calculen?

  Solució: Calculen x^y.

- Quina és més eficient i per què?

  Solució: `puzzle1` és més eficient. La complexitat de `puzzle1` és $O(\log y)$ mentre que la complexitat de `puzzle2` és $O(y)$. Per exemple, per a calcular x^{64}, l'algorisme de `puzzle2` fa 32 multiplicacions de x^2. L'algorisme `puzzle1` calcula $x^2 = x \cdot x$, $x^4 = x^2 \cdot x^2$, $x^8 = x^4 \cdot x^4$, etc, i així fa un nombre logarítmic de multiplicacions, aprofitant els càlculs fets a les crides anteriors.

16. Dissenyar una funció amb l'especificació següent:

```
// Pre: n >= 0
// Retorna cert si la suma dels digits a les posicions parells es igual
// a la suma dels digits a les posicions senars, suposant que el nombre
// esta representat en base 10. En cas contrari retorna fals.

bool mateixaSuma(int n);
```

Restricció: El codi de la funció no pot fer servir cap instrucció ‘‘`if`’’ ni cridar a cap altra funció.

Solució:

```
bool mateixaSuma(int n) {
  int sum = 0;
  int sign = 1;
  while (n > 0) {
    sum += (n%10)*sign;
    sign = -sign;
    n = n/10;
  }
  return sum == 0;
}
```

Matrius

17. **Patrons:** Volem dissenyar una funció anomenada `patro`, seguint l'esquema proposat més avall, per detectar si una matriu quadrada M té un patró determinat només observant si els elements són iguals o diferents de zero. Cal definir una expressió que substitueixi `Condició` per tal que la funció detecti el patró de l'esquerra, tenint en compte que els elements representats per una caixa negra (■) representen valors diferents de zero.
Important: En aquest problema es penalitzarà la utilització dels operadors `and` i `or`. Cal proposar expressions que només facin servir operadors aritmètics ($+, -, *, /, \%$) o relacionals (`==`, `!=`, `<`, `<=`, ...). Per exemple, per saber si una matriu és diagonal, amb tots els elements de la diagonal diferents de zero i la resta zero, es podria fer servir la condició següent:

```
(i == j) == (M[i][j] == 0)
```

Nota: Suposeu que l'element `M[0][0]` és el de la posició superior esquerra de la figura.

```
bool patro(const vector< vector<int> >& M) {
  int n = M.size();
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        if ( Condició ) return false;
    }
  }
  return true;
}
```

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \blacksquare \\ 0 & 0 & 0 & 0 & \blacksquare & 0 \\ 0 & 0 & 0 & \blacksquare & 0 & 0 \\ 0 & 0 & \blacksquare & 0 & 0 & 0 \\ 0 & \blacksquare & 0 & 0 & 0 & 0 \\ \blacksquare & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Solució: `(i + j == n - 1) == (M[i][j] == 0)`

$$M = \begin{bmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ 0 & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ 0 & 0 & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ 0 & 0 & 0 & \blacksquare & \blacksquare & \blacksquare \\ 0 & 0 & 0 & 0 & \blacksquare & \blacksquare \\ 0 & 0 & 0 & 0 & 0 & \blacksquare \end{bmatrix}$$

Solució: `(i <= j) == (M[i][j] == 0)`

$$M = \begin{bmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & 0 & 0 \\ \blacksquare & \blacksquare & \blacksquare & 0 & 0 & 0 \\ \blacksquare & \blacksquare & 0 & 0 & 0 & 0 \\ \blacksquare & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Solució: `(i + j < n) == (M[i][j] == 0)`

$$M = \begin{bmatrix} \blacksquare & 0 & \blacksquare & 0 & \blacksquare & 0 \\ 0 & \blacksquare & 0 & \blacksquare & 0 & \blacksquare \\ \blacksquare & 0 & \blacksquare & 0 & \blacksquare & 0 \\ 0 & \blacksquare & 0 & \blacksquare & 0 & \blacksquare \\ \blacksquare & 0 & \blacksquare & 0 & \blacksquare & 0 \\ 0 & \blacksquare & 0 & \blacksquare & 0 & \blacksquare \end{bmatrix}$$

Solució: `((i + j)%2 == 0) == (M[i][j] == 0)`

18. Considereu la funció següent, on A és una matriu quadrada que només conté elements enters positius:

```
int diagonalInferiorSuma(const matriu& A, int sum);
```

La funció retorna l'índex de la primera diagonal inferior de la matriu (incloent la diagonal principal) tal que la suma dels seus elements és exactament `sum`. Considerem que l'index d'una diagonal és la fila de l'element $A[i][0]$ de la diagonal.
A l'exemple següent:

$$A = \begin{bmatrix} 3 & 4 & 5 & 1 & 8 \\ \mathbf{1} & 2 & 9 & 7 & 2 \\ 6 & \mathbf{3} & 1 & 5 & 6 \\ \mathbf{5} & 1 & \mathbf{1} & 1 & 1 \\ 4 & \mathbf{8} & 2 & \mathbf{8} & 9 \end{bmatrix}$$

la crida `diagonalInferiorSuma(`A`, 13)` hauria de retornar 1, atès que els elements de la diagonal $(1, 3, 1, 8)$ sumen 13. També hi ha una altra diagonal que ho compleix $(5, 8)$, però té un índex superior a l'anterior. En el cas que no hi hagi cap diagonal inferior que sumi `sum`, la funció ha de retornar -1. Es valorarà principalment l'eficiència de la funció i la seva senzillesa.

Solució:

```
int diagonaInferiorSuma(const matriu& A, int sum) {
  int n = A.size();
  for (int i = 0; i < n; ++i) { // index de la diagonal
      int suma = 0;
      int j = 0;
      // Recorrem la diagonal i acabem tant aviat superem sum
      while (suma <= sum and (i+j) < n) {
        suma += A[i+j][j];
        ++j;
      }
      if (suma == sum) return i;
  }
  return -1;
}
```

19. Suposem que A és una matriu quadrada $n \times n$ inicialitzada tota a zeros. Volem omplir un triangle visitant els elements de la matriu per diagonals, tal com mostren les figures. Completeu els codis per tal d'omplir la matriu segons l'esquema que es mostra a la seva esquerra, tenint en compte que les files van numerades de dalt a baix $(0 \ldots n-1)$ i les columnes van numerades d'esquerra a dreta $(0 \ldots n-1)$.

$$A = \begin{bmatrix} 1 & 3 & 6 & 10 & 15 \\ 2 & 5 & 9 & 14 & \\ 4 & 8 & 13 & & \\ 7 & 12 & & & \\ 11 & & & & \end{bmatrix}$$

```
vector< vector<int> > A(n, vector<int>(n, 0));
int k = 1;
for (int i = 0; i < n; ++i) {
    for (int j = 0; j <= i; ++j) {
        A[i - j][j] = k;
        ++k;
    }
}
```

$$A = \begin{bmatrix} & & & & 5 \\ & & & 4 & 9 \\ & & 3 & 8 & 12 \\ & 2 & 7 & 11 & 14 \\ 1 & 6 & 10 & 13 & 15 \end{bmatrix}$$

```
vector< vector<int> > A(n, vector<int>(n, 0));
int k = 1;
for (int i = 0; i < n; ++i) {
    for (int j = i; j < n; ++j) {
        A[n - 1 + i - j][j] = k;
        ++k;
    }
}
```

20. **Matrius de Toeplitz**
Una matriu quadrada és de Toeplitz si els elements de les seves diagonals (descendents d'esquerra a dreta) són constants. Aquest és un exemple de matriu de Toeplitz:

$$\begin{bmatrix} 1 & 6 & 0 & 1 & -2 \\ -3 & 1 & 6 & 0 & 1 \\ 2 & -3 & 1 & 6 & 0 \\ 7 & 2 & -3 & 1 & 6 \\ -5 & 7 & 2 & -3 & 1 \end{bmatrix}$$

Dissenyeu una funció que ens digui si una matriu quadrada és de Toeplitz. La funció ha de tenir l'especificació següent:

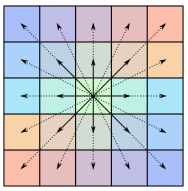
```
// M es una matriu quadrada no buida
// Retorna cert si la matriu es de Toeplitz
bool toeplitz(const vector<vector<int>>& M);
```

Solució: És suficient amb dissenyar una funció que analitzi parells de files consecutives i comprovi que la segona és un desplaçament de la primera.

```
bool toeplitz(const vector<vector<int>>& M) {
  int n = M.size();
  for (int i = 0; i < n - 1; ++i) {
    for (int j = 0; j < n - 1; ++j) {
      if (M[i][j] != M[i+1][j+1]) return false;
    }
  }
  return true;
}
```

21. **Matrius centrosimètriques:**
Una matriu quadrada és centrosimètrica si és simètrica en relació al seu centre. Per exemple, aquestes dues matrius són centrosimètriques:

$$\begin{bmatrix} 4 & 1 & 6 & 9 & 0 \\ 0 & 2 & 8 & 1 & 2 \\ 5 & 4 & 3 & 4 & 5 \\ 2 & 1 & 8 & 2 & 0 \\ 0 & 9 & 6 & 1 & 4 \end{bmatrix} \qquad \begin{bmatrix} 3 & 8 & 1 & 0 \\ 4 & 9 & 6 & 2 \\ 2 & 6 & 9 & 4 \\ 0 & 1 & 8 & 3 \end{bmatrix}$$



La matriu següent no és centrosimètrica, atès que els elements amb valors 5 i 4 són diferents:

$$\begin{bmatrix} 1 & 5 & 6 \\ 0 & 3 & 0 \\ 6 & 4 & 1 \end{bmatrix}$$

Però també podem tenir matrius rectangulars centrosimètriques:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 9 & 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 & 0 \end{bmatrix}$$

Dissenyeu una funció que digui si una matriu rectangular és centrosimètrica. La funció ha de tenir l'especificació següent:

```
// M es una matriu no buida
// Retorna cert si la matriu es centrosimetrica
bool centrosimetrica(const vector<vector<int>>& M);
```

Solució:

```
bool centrosimetrica(const vector<vector<int>>& M) {
  int n = M.size();
  int m = M[0].size();
  // Nomes cal mirar la meitat de la matriu
  // (potser repetim alguns elements de la fila del mig)
  for (int i = 0; i < (n+1)/2; ++i) {
    for (int j = 0; j < m; ++j) {
      if (M[i][j] != M[n-1-i][m-1-j]) return false;
    }
  }
  return true;
}
```

22. **Cerca binària en matriu**
Completeu la funció següent per tal que faci una cerca binària en un vector ordenat:

```
int CercaVector(const vector<int>& V, int x) {
  int i = 0;
  int j = V.size() - 1;
  while (i <= j) {
    int k =
    if (V[k] == x)
    if (V[k] < x)
    else
  }
  return -1;
}
```

Solució:

```
int CercaVector(const vector<int>& V, int x) {
  int i = 0;
  int j = V.size() - 1;
  while (i <= j) {
    int k = (i + j)/2;
    if (V[k] == x) ;
    if (V[k] < x) i = k + 1;
    else j = k - 1;
  }
  return -1;
}
```

Una matriu està completament ordenada si els elements de cada fila estan ordenats en ordre ascendent i el primer element de cada fila és més gran que l'últim element de la fila anterior. Per exemple, aquesta matriu està completament ordenada:

$$\begin{bmatrix} 1 & 3 & 4 & 5 \\ 7 & 9 & 11 & 15 \\ 16 & 20 & 21 & 23 \end{bmatrix}$$

Dissenyeu una funció que faci una cerca eficient en una matriu no buida completament ordenada segons la especificació següent:

```
void CercaMatriu(const vector<vector<int>>& M, int x,
                 int& fila, int& col);
```

La funció retorna la fila i columna on es troba l'element en els dos paràmetres per referència. En cas que no es trobi, retorna -1 en els dos paràmetres.
Pista: Imagineu la matriu com un vector molt llarg.

Solució:

```
void CercaMatriu(const vector<vector<int>>& M, int x,
                 int& fila, int& col) {
  int nfiles = M.size();
  int ncols = M[0].size();
  int i = 0;
  int j = nfiles*ncols - 1;
  while (i <= j) {
    int k = (i + j)/2;
    fila = k/ncols;
    col = k%ncols;
    if (M[fila][col] == x) return;
    if (M[fila][col] < x) i = k + 1;
    else j = k - 1;
  }
  fila = col = -1;
}
```

Pregunta: Quina complexitat té la funció que heu dissenyat suposant que la matriu té dimensió $m \times n$? Raoneu la resposta.

Solució: L'algorisme implementa una cerca dicotòmica sobre un vector de $n \times m$ elements. La complexitat és $O(\log nm)$. Això ho podem expressar com $O(\log n + \log m)$ o també $O(\log \max(n, m))$.

---

23. Suposem que A és una matriu $n \times m$, x és un vector de m elements i b és un vector de n elements. Dissenyeu la funció:

```
bool SistemaCorrecte(const vector<vector<int>>& A,
                     const vector<int>& x,
                     const vector<int>& b)
```

que retorna *cert* si $Ax = b$ i retorna *fals* en cas contrari.

Solució:

```
bool SistemaCorrecte(const vector<vector<int>>& A,
                     const vector<int>& x,
                     const vector<int>& b) {
  int n = A.size();
  int m = x.size();

  for (int i = 0; i < n; ++i) {
    int sum = 0;
    for (int j = 0; j < m; ++j) sum += A[i][j]*x[j];
    if (sum != b[i]) return false;
  }
  return true;
}
```

24. Suposem que A és una matriu quadrada i b és un vector amb tants elements com files té A. També sabem que A és una matriu triangular inferior, és a dir, $A[i][j] = 0$ quan $j > i$, i que tots els elements de la diagonal són diferents de zero.
Dissenyeu la funció següent que retorna un vector x tal que $Ax = b$:

```
vector<double> SistemaEquacions(const vector<vector<double>>& A,
                                const vector<double>& b)
```

Solució:

```
vector<double> SistemaEquacions(const vector<vector<double>>& A,
                                const vector<double>& b) {
  int n = b.size();
  vector<double> x(n);
  for (int i = 0; i < n; ++i) {
    double sum = 0;
    for (int j = 0; j < i; ++j) sum += A[i][j]*x[j];
    x[i] = (b[i] - sum)/A[i][i];
  }
  return x;
}
```

Responeu a les preguntes següents:

- Quina és la complexitat de l'algorisme que heu dissenyat?

  Solució: L'algorisme té complexitat $O(n^2)$, on n és la dimensió de la matriu.

- Imagineu que A és una matriu on només la diagonal i l'anti-diagonal són diferents de zero? Quina és la complexitat mínima que tindria un algorisme que resolgués el sistema d'equacions?

  Solució: Complexitat $O(n)$.

- Descriviu informalment (sense escriure el codi) com funcionaria l'algorisme anterior.

  Solució: L'algorisme només ha de resoldre $n/2$ sistemes de dues equacions amb dues incògnites amb les parelles de files $(0, n-1)$, $(1, n-2)$, etc. Cada sistema de dues equacions es pot resoldre en temps constant.

25. **Sistema d'inequacions:**
Considereu les definicions següents:

```
typedef vector<double> vec;  // vector de reals
typedef vector<vec> mat;     // matriu de reals
```

Dissenyeu una funció amb l'especificació següent:

```
// x i y son vectors amb el mateix nombre d'elements
// Retorna el producte escalar x*y
double ProducteEscalar(const vec& x, const vec& y);
```

Solució:

```
double ProducteEscalar(const vec& x, const vec& y) {
  double sum = 0;
  for (int i = 0; i < x.size(); ++i) sum += x[i]*y[i];
  return sum;
}
```

Considereu la funció següent, on A és una matriu $n \times m$, x és un vector de m elements i b és un vector de n elements:

```
bool SistemaIneq(const mat& A, const vec& x, const vec& b);
```

Dissenyeu la funció de tal manera que retorni *cert* si $Ax \leq b$, i *fals* en cas contrari. Es valorarà l'eficiència de la solució i la utilització de la funció anterior.

Solució:

```
bool SistemaIneq(const mat& A, const vec& x, const vec& b) {
  for (int i = 0; i < A.size(); ++i) {
      if (ProducteEscalar(A[i], x) > b[i]) return false;
  }
  return true;
}
```

Finalment, considereu que tenim un vector c de m elements i una matriu X de dimensions $p \times m$ on cada fila x_i conté un vector de m elements. Dissenyeu la funció següent:

```
int MaximVec(const mat& A, const mat& X,
             const vec& b, const vec& c);
```

de tal manera que retorni l'index i del vector x_i que maximitzi el producte escalar $c \cdot x_i$ i compleixi que $Ax_i \leq b$. Si cap vector x_i compleix el sistema d'inequacions, llavors cal retornar `-1`. Si n'hi ha més d'un, cal retornar-ne un d'ells.

Solució:

```
int MaximVec(const mat& A, const mat& X,
             const vec& b, const vec& c) {
  int idx = -1;
  int maxprod;
  for (int i = 0; i < X.size(); ++i) {
    if (SistemaIneq(A, x[i], b)) {
      double prod = ProducteEscalar(c, x[i]);
      if (idx < 0 or prod > maxprod) {
        idx = i;
        maxprod = prod;
      }
    }
  }
  return idx;
}
```

Quina complexitat té la funció `MaximVec`? Raoneu la resposta.

Solució: En el cas pitjor, `MaximVec` executa p vegades `SistemaIneq` i `ProducteEscalar`. Podem ignorar la complexitat de `ProducteEscalar` donat que és inferior a la de `SistemaIneq`. La complexitat de `SistemaIneq` és $(n \cdot m)$, donat que fa n productes escalars de vectors d'm elements. Per tant, la complexitat de `SistemaIneq` és $O(n \cdot m \cdot p)$.

Complexitat d'algorismes

26. Suposem que tenim un vector d'enters amb n elements i que el vector està ordenat de forma estrictament creixent. Donat un nombre x, volem saber si existeix alguna parella d'elements $V[i]$ i $V[j]$, amb $i \neq j$, tal que $V[i]+V[j] = x$. Proposeu tres algorismes diferents per a resoldre aquest problema que tinguin complexitat mínima $O(n^2)$, $O(n \log n)$ i $O(n)$, respectivament. Per a cada cas, descriviu molt breument i informalment com funcionaria l'algorisme. No cal escriure el codi.

- Algorisme amb complexitat $O(n^2)$

  Solució: L'algorisme té dos bucles imbricats, un recorrent el vector amb l'índex i i l'altre amb l'índex j. Per a cada parella $V[i]$ i $V[j]$ es comprova si la seva suma és x. L'algorisme s'atura quan troba la parella tal que $V[i]+V[j] = x$. En el pitjor dels casos visitarà totes les parelles d'elements (n^2).

- Algorisme amb complexitat $O(n \log n)$:

  Solució: L'algorisme té un bucle que recorre tot el vector. Per a cada element $V[i]$ es fa una cerca binària de l'element $x - V[i]$. En el pitjor dels casos, l'algorisme farà n cerques binàries, on cada cerca binària té complexitat $O(\log n)$.

- Algorisme amb complexitat $O(n)$:

  Solució: L'algorisme treballa amb dos índexos, i i j, on $i \leq j$. Els índexos indentifiquen un interval del vector on es pot trobar la parella d'elements. Inicialment, $i = 0$ i $j = n - 1$. L'algorisme té un bucle que a cada iteració mou i cap a la dreta o j cap a l'esquerra, depenent de si $V[i] + V[j]$ és més petit o més gran que x, respectivament. L'algorisme acaba quan $V[i] + V[j] = x$ o quan i i j coincideixen. El bucle fa com a màxim n iteracions.

27. Diem que un vector V de n elements és una *muntanya* si existeix un element p, amb $0 < p < n-1$, tal que $V[0 \dots p]$ està ordenant en ordre estrictament creixent i $V[p \dots n-1]$ en ordre estrictament decreixent. Per exemple, aquest és un vector muntanya:

$$V = [2, 5, 6, 8, 10, 12, 13, 15, 14, 11, 10, 9, 7, 6, 5, 3, 1, 0]$$

Donat un vector muntanya, digueu quina complexitat tindrien algorismes eficients per resoldre els problemes següents. Per a cada cas, descriviu molt breument i informalment com funcionaria l'algorisme. No cal escriure el codi.

- Trobar el cim de la muntanya.

  Solució: Complexitat $O(\log n)$. L'algorisme fa una cerca dicotòmica de l'element tal que els seus dos veins són més petits que ell mateix. Durant la cerca eliminarem la meitat dels elements a cada iteració depenent de si estem a la zona creixent o a la zona decreixent.

- Ordenar el vector.

  Solució: Complexitat $O(n)$. L'algorisme primer ha de trobar el cim p del vector ($O(\log n)$) i després executar un algorisme semblant a la fusió de dos vectors ordenats, $V[0 \dots p-1]$ i $V[p \dots n-1]$. La fusió té complexitat $O(n)$.

- Cercar un element en el vector.

  Solució: Complexitat $O(\log n)$. L'algorisme primer ha de trobar el cim p del vector ($O(\log n)$) i després fer dues cerques dicotòmiques, una a la part creixent i l'altra a la part decreixent. Cada cerca té complexitat $O(\log n)$, com a màxim.

- Trobar l'element repetit més gran del vector. A l'exemple, l'element més gran repetit és el 10.

  Solució: Complexitat $O(n)$. Des de el cim del vector cal anar baixant per la part creixent i decreixent, sempre avançant per l'element més gran. L'algorisme s'aturaria quan trobés dos element iguals o quan s'hagués recorregut tot el vector. En el pitjor dels casos faria n iteracions.

28. **Matriu ordenada:** Sigui A una matriu quadrada $n \times n$. La matriu està ordenada per files i columnes, és a dir,

    `A[i][j] <= A[i+1][j]` i `A[i][j] <= A[i][j+1]`

    Exemple de matriu ordenada:

$$A = \begin{bmatrix} 1 & 4 & 5 & 7 & 10 & 12 \\ 2 & 5 & 8 & 9 & 10 & 13 \\ 6 & 7 & 10 & 11 & 12 & 15 \\ 9 & 11 & 13 & 14 & 17 & 20 \\ 11 & 12 & 19 & 20 & 21 & 23 \\ 13 & 14 & 20 & 22 & 25 & 26 \end{bmatrix}$$

    Indiqueu la complexitat, amb notació $O()$, que tindrien els càlculs següents i descriviu, de manera informal, l'estratègia que es faria servir. Per a cada càlcul cal proposar l'algorisme més eficient possible.

    - Cercar un element x dins de la matriu:

      Solució: Complexitat $O(n)$. L'algorisme està explicat al material del curs. Comença per una de les cantonades de la matriu (per exemple, la inferior esquerra) i va eliminant una fila o columna a cada iteració. En el pitjor dels casos es faran $2n$ iteracions.

    - Cercar una fila tal que els seus elements sumin un valor donat x:

      Solució: Complexitat $O(n \log n)$. Cal fer una cerca binària sobre les files de la matriu ($\log_2 n$ iteracions) tenint en compte que les seves sumes també estan ordenades. Per a cada iteració cal calcular la suma dels elements de la fila ($O(n)$).

    - Cercar l'element més gran de la matriu:

      Solució: Complexitat $O(1)$. És l'element `A[n-1][n-1]`.

    - Mirar si tots els elements de la matriu són iguals:

      Solució: Complexitat $O(1)$. L'element més gran està a `A[n-1][n-1]` i l'element més petit a `A[0][0]`. Només cal mirar que `A[0][0] == A[n-1][n-1]`.

    - Sumar tots els elements de la matriu:

      Solució: Complexitat $O(n^2)$. Cal visitar tots els elements i acumular la seva suma en una variable.

    - Calcular $A^2 = A \times A$:

      Solució: Complexitat $O(n^3)$. Cal utilitzar l'algorisme clàssic de multiplicació de matrius amb tres bucles imbricats.

- Donades dues matrius ordenades A i B amb tots els seus elements positius, saber si $A \times B$ és una matriu ordenada:

  Solució: Complexitat $O(1)$. No cal fer cap càlcul donat que la multiplicació de matrius ordenades amb tots els elements positius és sempre una matriu ordenada. La demostració és senzilla: cal comprovar que els veïns de cada element estan ordenats.

- Calcular A^n, on n és el nombre de files/columnes de la matriu:
  (per simplificar l'anàlisi de complexitat, podeu suposar que n és una potència de 2)

  Solució: Complexitat $O(n^3 \log n)$. Suposem que n és una potència de 2. Llavors anem calculant successivament $A^2 = A \times A$, $A^4 = A^2 \times A^2$, $A^8 = A^4 \times A^4$, ..., fins arribar a A^n. Amb aquesta estratègia cal fer $\log_2 n$ multiplicacions de matrius.

---

29. **Complexitat:** Suposeu que la funció `something(`i`)` té complexitat $O(i)$. Quina és la complexitat de cadascuna de les funcions següents? Raoneu breument les respostes explicant quin càlcul heu fet per arribar a l'expressió de la complexitat.

```
void f(int n) {
  if (n == 0) return;
  f(n/2);
  something(1);
  f(n/2);
}
```

Solució:

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) = 2 \cdot 2 \cdot T(n/4) \\ &= 2 \cdot 2 \cdots 2 \qquad (\log_2 n \text{ vegades}) \\ &= 2^{\log_2 n} = n \end{aligned}$$

La complexitat és $O(n)$.

```
void g(int n) {
  for (int i=0; i<n; ++i) {
    something(i);
  }
}
```

Solució:

$$T(n) = 0 + 1 + 2 + \cdots + n - 1 = n(n-1)/2$$

La complexitat és $O(n^2)$.

```
void h(int n) {
  if (n==0) return;
  h(n/2);
  something(n);
  h(n/2);
}
```

Solució:

$$\begin{aligned} T(n) &= n + 2 \cdot T(\tfrac{n}{2}) = n + 2\left(\frac{n}{2} + 2 \cdot T(\tfrac{n}{4})\right) \\ &= n + n + \cdots + n \qquad (\log_2 n \text{ vegades}) \end{aligned}$$

La complexitat és $O(n \log n)$.

```
void p(int n) {
  if (n==0) return something(1);
  for (int i=0; i<n; ++i) p(i);
}
```

Solució:

$$\begin{aligned} T(n) &= T(0) + T(1) + \cdots + T(n-1) \\ T(n+1) &= 2 \cdot T(n) \end{aligned}$$

Per tant:

$$T(n) = 2^0 + 2^1 + 2^2 + \cdots + 2^{n-1} = 2^n - 1$$

La complexitat és $O(2^n)$.

30. En aquest exercici farem servir lletres majúscules (A, B) per representar matrius d'$n \times n$ i lletres minúscules (a, b) per representar vectors d'n elements.
Considerant els algorismes que hem vist durant el curs per a vectors i matrius, digueu quina és la menor complexitat assimptòtica que podríeu aconseguir per a realitzar les operacions següents (raoneu la resposta en els casos que es demana):

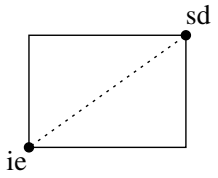
	Complexitat
$a \cdot b$	$O(n)$
$A \cdot b$	$O(n^2)$
$A \cdot B$	$O(n^3)$
A^T	$O(n^2)$

	Complexitat	**Raonament**
$A \cdot x = b$	$O(n^3)$	Es pot realitzar amb una eliminació Gaussiana que té tres bucles imbricats.
A^{-1}	$O(n^3)$	Es pot realitzar amb una eliminació Gaussiana per resoldre n sistemes d'equacions. Té la mateixa complexitat que $A \cdot x = b$.
A^n	$O(n^3 \log n)$	Fent servir el producte eficient (nombre logarítmic de productes), es pot fer amb $\log n$ multiplicacions de matrius.
$A^n \cdot b$	$O(n^3)$	Es pot fer amb n multiplicacions de matriu per vector, començant per la dreta: $(A \dots (A(A(A \cdot b))) \dots)$.

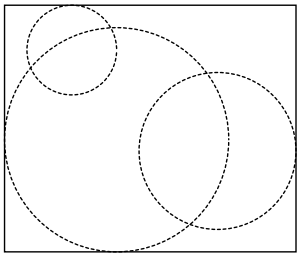
Monte Carlo

31. **Cercles:** Considereu les següents definicions de tipus:

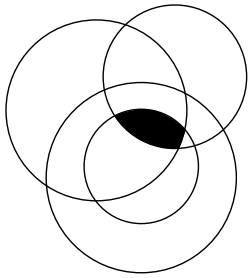
```
struct Punt {
  double x, y;
};

struct Ortoedre {
  Point iep, sda;
};

struct Esfera {
  Point c;
  double r;
};
```



(a)



(b)



(c)

Un punt a R^2 té dues coordenades (x, y). Un rectangle és rectilini quan té les arestes paral.leles als eixos. Un rectangle rectilini es pot representar amb dos punts, els extrems inferior-esquerre (ie) i superior-dret (sd) de la diagonal, tal com mostra la figura (a). Un cercle es representa amb el centre (c) i el radi (r).
Dissenyeu les funcions següents segons la seva especificació:

- Una funció que diu si un punt es troba dins d'un cercle. Podeu suposar que els punts de la circumferència estan dins del cercle.

Solució:

```
// Mirar si la distancia al centre es menor que el radi
bool inside(const Point& P, const Circle& O) {
  double dx = P.x - O.c.x;
  double dy = P.y - O.c.y;
  return dx*dx + dy*dy <= O.r*O.r;
}
```

- Una funció que retorna el rectangle rectilini més petit que conté tots els cercles del vector, tal com mostra l'exemple de la figura (b). Podeu suposar que el vector no és buit i podeu fer servir les funcions `min` i `max`.

Solució:

```
Rectangle BoundingBox(const VCircle& V) {
  Rectangle BB;
  BB.ie = BB.sd = V[0].c;
  for (int i = 0; i < V.size(); ++i) {
    BB.ie.x = min(BB.ie.x, V[i].c.x - V[i].r);
    BB.ie.y = min(BB.ie.y, V[i].c.y - V[i].r);
    BB.sd.x = max(BB.sd.x, V[i].c.x + V[i].r);
    BB.sd.y = min(BB.sd.y, V[i].c.y + V[i].r);
  }
  return BB;
}
```

- Una funció que retorna un punt aleatori dins del rectangle rectilini seguint una distribució uniforme. Podeu utilitzar la funció `rand()` que retorna un nombre enter a l'interval $[0, \texttt{RAND\_MAX})$.

Solució:

```
Point RandomPoint(const Rectangle& R) {
  Point P;
  P.x = (R.sd.x-R.ie.x)*rand()/RAND_MAX + R.ie.x;
  P.y = (R.sd.y-R.ie.y)*rand()/RAND_MAX + R.ie.y;
  return P;
}
```

- Una funció que retorna una estimació de l'àrea de la intersecció de tots els cercles del vector, tal com mostra la figura (c). Utilitzeu les funcions dels apartats anteriors i el mètode de Monte Carlo, generant n punts aleatoris (podeu suposar que $n > 0$).

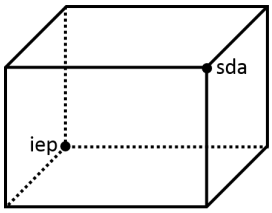
Solució:

```
double Area(const VCircle& V, int n) {
  Rectangle BB = BoundingBox(V);
  int nin = 0;
  for (int i = 0; i < n; ++i) {
    Point P = RandomPoint(BB);
    bool inside_all = true;
    int c = 0;
    while (inside_all and c < V.size()) {
      inside_all = inside(P, V[i]);
      ++c;
    }
    if (inside_all) ++nin;
  }
  double areaBB = (BB.sd.x - BB.ie.x)*(BB.sd.y - BB.ie.y);
  return areaBB*nin/n;
}
```

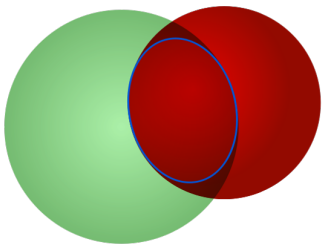
32. **Intersecció de dues esferes**

Considereu les següents definicions de tipus:

```
struct Punt {
  double x, y, z;
};

struct Ortoedre {
  Punt iep, sda;
};

struct Esfera {
  Punt c;
  double r;
};
```



(a)


(b)

Un punt a R^3 té tres coordenades (x, y, z). Un ortoedre és rectilini si té les cares paral·leles als eixos. Un ortoedre rectilini es pot representar amb dos punts, els extrems inferior-esquerre-posterior (iep) i superior-dret-anterior (sda) de la diagonal, tal com mostra la figura (a). Una esfera es representa amb el centre (c) i el radi (r).
Dissenyeu les funcions següents segons la seva especificació:

- Una funció que diu si un punt es troba en el volum inclòs dins de l'esfera.

Solució:

```
// Mirem si la distancia al centre es menor
// o igual que el radi
bool Dins(const Punt& P, const Esfera& E) {
  double dx = P.x - E.c.x;
  double dy = P.y - E.c.y;
  double dz = P.z - E.c.z;
  return dx*dx + dy*dy + dz*dz <= E.r*E.r;
}
```

- Una funció que retorna l'ortoedre rectilini més petit que conté completament dues esferes. Podeu fer servir les funcions `min` i `max`.

Solució:

```
Ortoedre BoundingBox(const Esfera& E1, const Esfera& E2) {
  Ortoedre BB;
  BB.iep.x = min(E1.c.x - E1.r, E2.c.x - E2.r);
  BB.iep.y = min(E1.c.y - E1.r, E2.c.y - E2.r);
  BB.iep.z = min(E1.c.z - E1.r, E2.c.z - E2.r);
  BB.sda.x = max(E1.c.x + E1.r, E2.c.x + E2.r);
  BB.sda.y = max(E1.c.y + E1.r, E2.c.y + E2.r);
  BB.sda.z = max(E1.c.z + E1.r, E2.c.z + E2.r);
  return BB;
}
```

- Una funció que retorna un punt aleatori, seguint una distribució uniforme, dins de l'ortoedre rectilini. Podeu utilitzar la funció `rand()` que retorna un nombre enter a l'interval $[0, \texttt{RAND\_MAX})$.

Solució:

```
Punt PuntAleatori(const Ortoedre& O) {
  Punt P;
  P.x = (O.sda.x-O.iep.x)*rand()/RAND_MAX + O.iep.x;
  P.y = (O.sda.y-O.iep.y)*rand()/RAND_MAX + O.iep.y;
  P.z = (O.sda.z-O.iep.z)*rand()/RAND_MAX + O.iep.z;
  return P;
}
```

- Una funció que retorna una estimació del volum de la intersecció de dues esferes, tal com mostra la figura (b). Utilitzeu les funcions dels apartats anteriors i el mètode de Monte Carlo, generant n punts aleatoris (podeu suposar que $n > 0$).

Solució:

```
double Volum(const Esfera& E1, const Esfera& E2, int n) {
  Ortoedre BB = BoundingBox(E1, E2);
  int nin = 0;
  for (int i = 0; i < n; ++i) {
    Punt P = PuntAleatori(BB);
    if (Dins(P, E1) and Dins(P, E2)) ++nin;
  }
  double volumBB = (BB.sda.x-BB.iep.x)*(BB.sda.y-BB.iep.y)*
                   (BB.sda.z-BB.iep.z);
  return volumBB*nin/n;
}
```

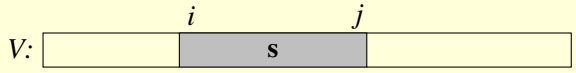
Diversos

33. **Invariants:**
Considereu la funció següent:

```
int SegmentSuma(const vector<int>& V, int sum);
```

on V és un vector no buit d'enters estrictament positius (no hi ha ni zeros ni negatius). La funció ha de cercar un segment del vector $V[i \dots j]$ tal que la suma dels seus elements sigui igual a `sum` (suposem que `sum` > 0). La funció ha de retornar l'índex i del primer element del segment. En cas que hi hagi més d'un segment que ho compleixi, ha de retornar el que té menor índex. En cas que no n'hi hagi cap, ha de retornar `-1`.
Exemple: Suposem el vector $V = [1\ 3\ 5\ 2\ 8\ 4\ 1\ 7\ 2]$.
Per `sum=10` la funció ha de retornar `1`, ja que 3+5+2=10.
Per `sum=14` la funció ha de retornar `3`, ja que 2+8+4=14.
Per `sum=6` la funció ha de retornar `-1`.
Es valorarà la correctesa, senzillesa i claretat del codi de la funció. Heu de trobar un algorisme amb **complexitat lineal** seguint les passes següents.
a) Dissenyeu un invariant pel bucle principal de l'algorisme, fent un dibuix i explicant breument el seu significat.

Solució: (hi pot haver altres solucions tan bones com aquesta)



s és la suma dels elements del segment $V[i \dots j]$ i no s'ha trobat cap segment que comenci per un índex inferior a i.

b) Indiqueu quina hauria de ser la condició d'**acabament** del bucle (while) segons l'invariant. Calculeu també la condició de **continuació** del bucle.

Solució:

```
Acabament:    j == V.size()-1 or s == sum
Continuació:  j != V.size()-1 and s != sum
```

c) Indiqueu quin hauria de ser el codi després de finalitzar el bucle principal.

Solució:

```
if (s == sum) return i;
else return -1;
```

d) Indiqueu com hauria de ser la inicialització del bucle.

Solució:
Dues opcions:

```
int i = 0, j = 0, s = V[0];
int i = 0, j = -1, s = 0;
```

e) Finalment, dissenyeu el cos del bucle (només la part de dins del while).

Solució:

```
if (s > sum) {
  s -= V[i];
  ++i;
} else {
  ++j;
  s += V[j];
}
```

---

34. Dissenyar una funció amb l'especificació següent:

```
// Pre: n >= 0
// Retorna cert si la suma dels digits a les posicions parells es igual
// a la suma dels digits a les posicions senars, suposant que el nombre
// esta representat en base 10. En cas contrari retorna fals.

bool mateixaSuma(int n);
```

Restricció: El codi de la funció no pot fer servir cap instrucció ‘‘`if`’’ ni cridar a cap altra funció.

Solució:

```
bool mateixaSuma(int n) {
  int sum = 0;
  int sign = 1;
  while (n > 0) {
    sum += (n%10)*sign;
    sign = -sign;
    n = n/10;
  }
  return sum == 0;
}
```

35. **Segment dins d'un interval**
Sigui la funció següent:

```
int SegmentInterval(const vector<int>& V, int inf, int sup);
```

on `V` és un vector no buit de nombres estrictament positius i els paràmetres compleixen:

$$0 < \texttt{inf} \leq \texttt{sup}$$

La funció retorna un índex j tal que existeix un segment de `V` que comença a la posició j i acaba en una posició k i la suma dels elements del segment valen un valor entre `inf` i `sup`, és a dir,

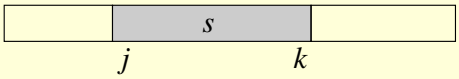
$$\texttt{inf} \leq \sum_{i=j}^{k} \texttt{V}[i] \leq \texttt{sup}.$$

En el cas que no existeixi cap segment que compleixi aquesta condició, la funció retorna `-1`. En el cas que hi hagi més d'un segment, retorna el que té el menor valor de j.
Exemples:

- `SegmentInterval([3, 11, 9, 4, 5, 4, 3, 5], 15, 17)` $= 3$
- `SegmentInterval([3, 11, 9, 4, 5, 4, 3, 5], 15, 21)` $= 1$
- `SegmentInterval([2, 4, 8, 4, 2, 6, 8, 4], 7, 7)` $= -1$

Descriviu un invariant per un algorisme iteratiu de la funció. La descripció ha de consistir en un dibuix del vector i una descripció del significat de **totes** les variables que intervenen en el bucle. Es valorarà principalment l'eficiència i la simplicitat de l'algorisme.

Solució:



Invariant:

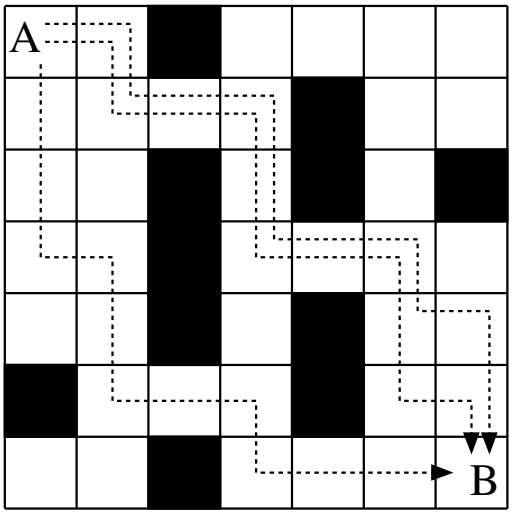
- j i k defineixen els extrems del segment que s'està analitzant.
- s és la suma dels elements en el segment `V[`$j \ldots k$`]`.
- No hi ha cap segment vàlid que comenci abans de la posició j.

Dissenyeu el codi de la funció segons l'invariant que heu proposat a l'apartat anterior.

Solució:

```
int SegmentInterval(const vector<int>& V, int inf, int sup) {
  int j = 0;
  int k = 0; // Comencem nomes amb el primer element
  int s = V[0];
  while (k < V.size()) {
    if (s < inf) {
      ++k;
      if (k < V.size()) s += V[k];
    } else if (s > sup) {
      s -= V[j];
      ++j;
    } else return j;
  }

  return -1;
}
```

36. **Camins reticulars:** Donat un reticle com el que mostra la figura (esquerra), volem calcular tots els camins reticulars que van de la casella `A` a la casella `B`. Un camí reticular només pot fer moviments a la dreta o avall, sense passar per cap dels obstacles del reticle (caselles negres). La figura mostra tres camins reticulars diferents.



15	6	0	9	4	4	0
9	6	5	5	0	4	0
3	1	0	5	0	4	0
2	1	0	5	4	4	1
1	1	0	1	0	3	1
0	1	1	1	0	2	1
0	0	0	1	1	1	1

- Calculeu quants camins reticulars diferents hi ha des de la casella `A` fins la `B`. Expliqueu informalment l'algorisme. Podeu fer servir el reticle de la dreta per fer anotacions i donar la solució. **Pista:** començar per la casella B.

  Solució: Hi ha 15 camins. Cal calcular els camins començant per la casella `B` (1 camí) i anar omplint en direcció a la casella `A` fent la suma dels dos veïns (dreta i avall). Per les caselles obstacle cal suposar que hi ha zero camins.

- Dissenyeu la funció següent:

```
typedef vector< vector<bool> > Reticle;
int numCamins(const Reticle& R);
```

  La funció rep un reticle rectangular (matriu booleana) on les caselles amb valor `fals` representen obstacles i les que tenen valor `cert` representen caselles lliures. La funció ha de retornar el nombre de camins reticulars que van de la casella `A` (`R[0][0]`) a l'altre extrem del reticle (`B`). Podeu suposar que les caselles `A` i `B` estan lliures. S'espera que l'algorisme sigui de complexitat polinòmica en la mida del reticle.

Solució:

```
int numCamins(const Reticle& R) {
  int n = R.size();
  int m = R[0].size();

  // Reticle amb comptador de camins (inicialment tot zeros).
  // Les caselles amb obstacles no les tocarem.
  vector< vector<int> > C(n, vector<int>(m,0));

  // Ultima fila
  C[n-1][m-1] = 1;
  for (j = m - 2; j >= 0; --j) {
    if (R[n-1][j]) C[n-1][j] = C[n-1][j+1];
  }

  // La resta del reticle (files n-2 a 0)
  for (i = n - 2; i >= 0; --i) {
    // Casella de mes a la dreta
    if (R[i][m-1]) C[i][m-1] = C[i+1][m-1];

    // La resta de caselles de la fila
    for (j = m - 2; j >= 0; --j) {
      if (R[i][j]) C[i][j] = C[i+1][j] + C[i][j+1];
    }
  }
  return C[0][0];
}
```