Recursivitat: introducció

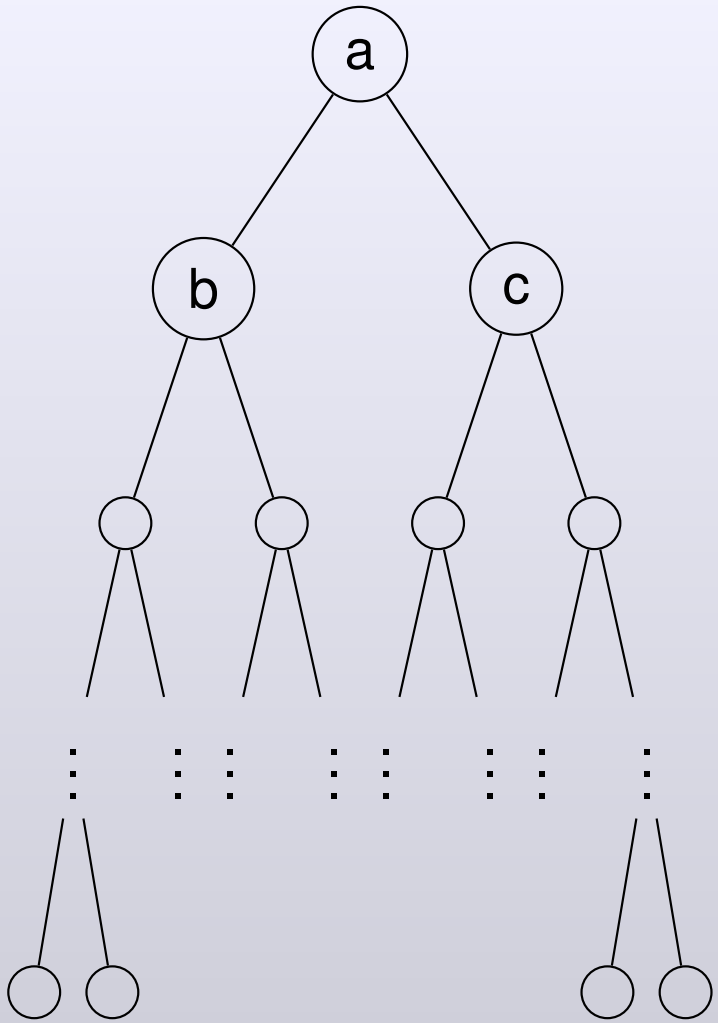
Recursivitat és la forma de definir un concepte en termes del mateix concepte que s'està definint

- Els nombres naturals ($\mathbb{N}$) es poden definir:
  - $0 \in \mathbb{N}$
  - si un nombre $n \in \mathbb{N}$ llavors el següent $n+1 \in \mathbb{N}$
- El factorial d'un nombre natural es pot definir:
  - $0! = 1$
  - si $n > 0$, $n! = n * (n-1)!$
- Si $x \in \mathbb{R}, x \neq 0$ i $n \in \mathbb{N}$ es defineix la potència n-èssima de x

Característiques de problemes recursius

- Es poden redefinir en termes de subproblemes del tipus inicial, però de “mida” menor
- Un dels subproblemes té solució directa o trivial (no recursiva)
- Aplicant la redefinició en termes dels subproblemes “petits” successivament s’arriba a problemes de solució trivial
- La solució dels problemes simples s’usa per a construir la solució del problema inicial

Exemple: Com recòrrer l'arbre que comença a "a"?



- passant per "a"
- recorrent l'arbre que comença a "b"
- recorrent l'arbre que comença a "c"

Funcions recursives

Una funció recursiva és una funció que té crides a ella mateixa

L'estructura d'aquestes funcions ha de contemplar:

- casos senzills: cal detectar-los i resoldre'ls directament
- casos recursius: es resolen amb casos més simples
- finalització: cal raonar que la funció acabará, indicant què es fa més petit en cada crida recursiva

Exemples: potència

- recursivament

```
double pot (double x, int n) {
  if ( n == 0 )
    return 1.;
  return x*pot(x,n-1);
}
```

- iterativament

```
double pot (double x, int n) {
  double p = 1;
  int i;
  for (i = 1; i <= n; i++)
    p *= x;
  return p;
}
```

Exemples: factorial

- recursivament

```
int fac (int n) {
  if ( n == 0 )
    return 1;
  return n*fac(n-1);
}
```

- iterativament

```
int fac (int n) {
  int f = 1, i;
  for (i = 1; i <= n; i++)
    f *= i;
  return f;
}
```

Exemple:màxim comú divisor

Algorisme d'Euclides: $\text{mcd}(0, b) = b$; si $a \neq 0$ $\text{mcd}(a, b) = \text{mcd}(b\%a, a)$

- recursivament

```
int mcd (int m, int n) {
  if ( m == 0 )
    return n;
  return mcd(n%m,m);
}
```

- iterativament

```
int mcd (int m, int n) {
  int k;
  while ( m != 0 ) {
    k = m; m = n%m; n = k;
  }
  return n;
}
```

Recursivitat indirecta:dues funcions mutuament recursives

- codi

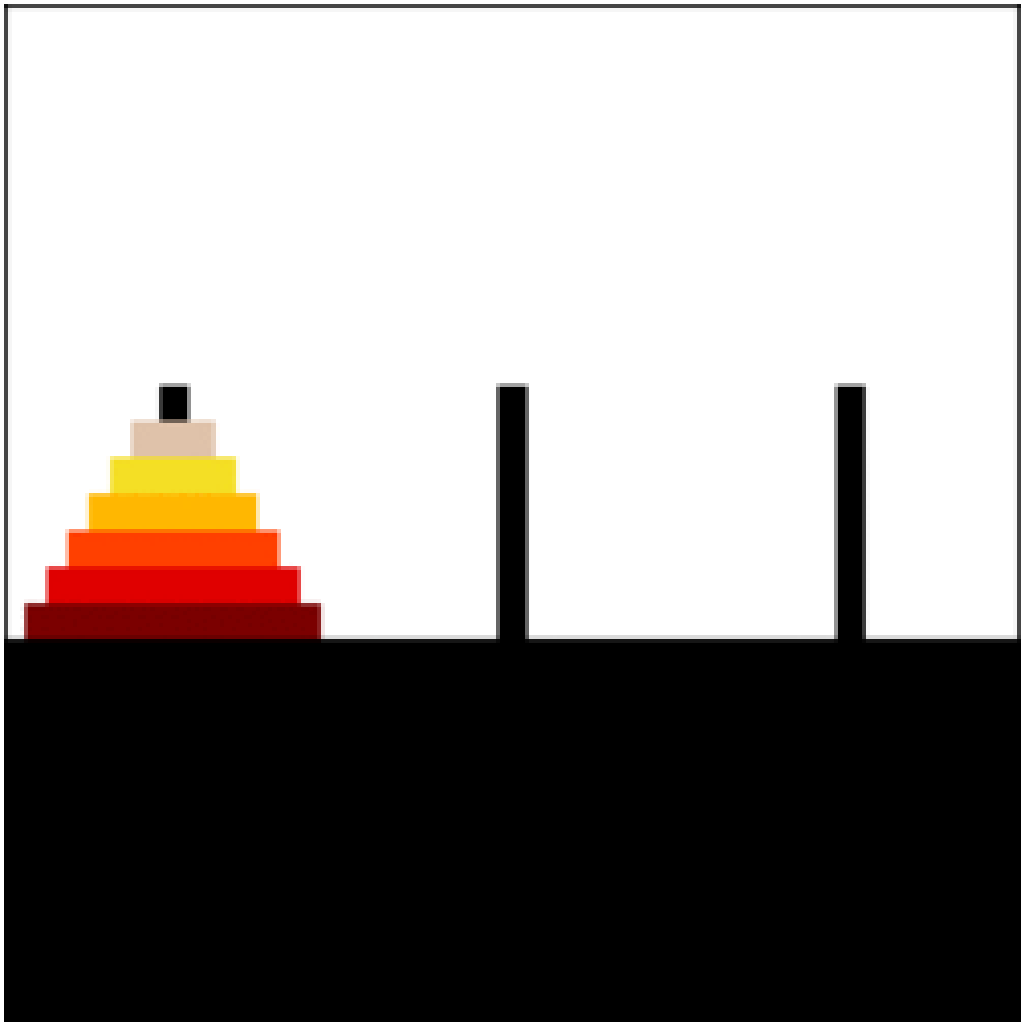
```c
#include <stdio.h>
void f1(char );
void f2(char );
int main(void) {
  f1('Z');
  printf("\n");
  return 0;
}
```

```c
void f1(char c) {
  if (c > 'A') f2(c);
  printf("%c ", c);
  return;
}
void f2(char c) {
  f1(c-1);
  return;
}
```

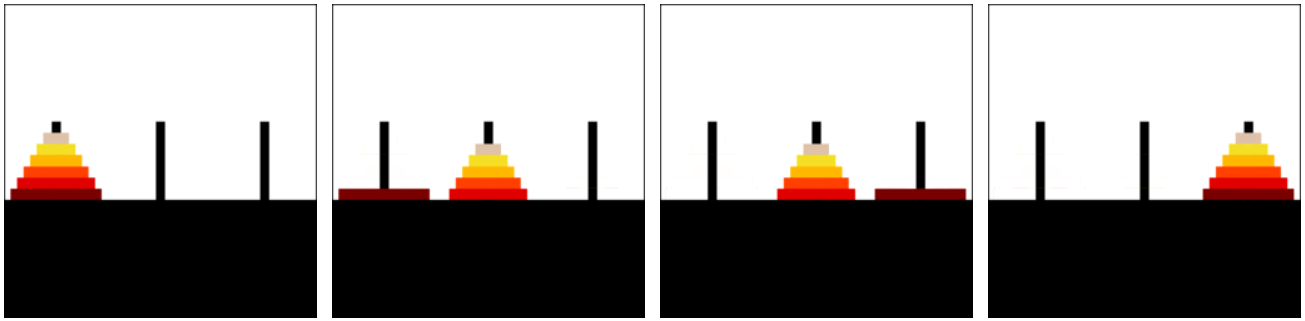
- Sortida

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Torres de Hanoi



Torres de Hanoi



Torres de Hanoi: `main`

```
#include <stdio.h>
void hanoi(char, char, char, int);

int main(void) {
  int n;
  printf("n = ? (>0) \n");
  scanf("%d", &n);

  if (n < 1)
    printf("dada incorrecta \n");
  else
    hanoi('E', 'D', 'C', n);
  return 0;
}
```

Torres de Hanoi: `hanoi` recursivament

```
void hanoi(char ini, char fin, char aux, int n) {
  if (n == 1)
    printf("moure de %c a %c\n", ini, fin);
  else {
    hanoi(ini, aux, fin, n-1);
    hanoi(ini, fin, aux, 1);
    hanoi(aux, fin, ini, n-1);
   }

  return;
}
```

Torres de Hanoi: iterativament

```
#include <stdio.h>

int main(void) {
  int orig[4100], dest[4100], n, i, j, i2;
  char nom[] = { 'E', 'C', 'D'};
  scanf("%d", &n);
  if ( n < 1 || n > 12) {
    printf("dada␣incorrecta\n");
    return 0;
  }
  orig[1] = 0;
  dest[1] = 2;
  i2 = 1; /* i2= 2^i */
```

Torres de Hanoi: iterativament

```
    for (i = 2; i <= n; i++) {
      i2 *= 2;
      for (j = 1; j < i2; j++) {
        orig[j] = (3 - orig[j]) % 3;
        dest[j] = (3 - dest[j]) % 3;
      }
      orig[i2] = 0; dest[i2] = 2;
      for (j = 1; j < i2; j++) {
        orig[i2+j] = (orig[j] + 1) % 3;
        dest[i2+j] = (dest[j] + 1) % 3;
      }
    }
    for (i = 1; i < 2 * i2; i++)
      printf("moure de %c a %c\n",
        nom[orig[i]], nom[dest[i]]);
    return 0;
}
```

Determinant: recursivament

```
double determinant (int m, double **a) {
  double sum = 0., **new, sign=1;
  int i,j,k;

  /* Casos simples */
  if (m == 1) return a[0][0];
  if (m == 2)
    return (a[0][0]*a[1][1] - a[0][1]*a[1][0]);

  /* Espai per a matriu de dimensio m-1 */
  new=(double **) malloc((m-1)*sizeof(double *));
  for (i=0; i< m-1; i++)
    new[i]=(double *) malloc((m-1)*sizeof(double));
```

Determinant: recursivament (cont.)

```
  for (i = 0; i < m; i++) {
/* es suprimeix fila 0 i columna "i" */
    for (j = 1; j < m; j++)
      for (k = 0; k < i; k++)
        new[j-1][k] = a[j][k];
      for (k = i+1; k < m; k++)
        new[j-1][k-1] = a[j][k];
/*  producte del determinant adjunt per
  a[0][i] amb signe adequat */
    sum += a[0][i]*sign*(determinant (m - 1, new));
    sign=-sign;
  }
  for (i=0; i< m-1; i++) free(new[i]);
  free(new);
  return sum;
}
```

Determinant: `main`

```
int main(void) {
  double **a, det;
  int i,j,n,k;
  do {
    scanf("%d",&n);
  } while (n<2);
  a = (double**) malloc(n*sizeof(double*));
  if (a == NULL){
     perror("malloc"); exit(2);
  }
  for(i=0;i<n;i++){
    a[i] = (double*) malloc(n*sizeof(double));
    if (a[i] == NULL){
       perror("malloc"); exit(2);
    }
  }
```

Determinant: `main` (cont.)

```
  for (i=0;i<n;i++) {
    for (j=0;j<n;j++)
    scanf("%le", &a[i][j]);
  }

  det=determinant(n,a);
  printf("␣%15.7e\n",det);

  for(i=0;i<n;i++) free (a[i]);
  free(a);
  return 0;
}
```

Determinant: recurrentment (Gauss pivotatge)

Atenció: destrueix la matriu

```
double determinant(double **a, int n){
  int i, j, k;
  double *aux, piv, det=1.;

  for(i = 0; i < n; i++){
    /* Cerquem modul maxim entre
       a[i][i], .... , a[n-1][i] */
    k = i;
    piv = fabs(a[i][i]);
    for(j = i+1; j < n; j++)
      if (fabs(a[i][j]) > piv){
        piv = fabs(a[i][j]);
        k = j;
      }
```

Determinant: recurrentment (cont.)

```
/* tota la columna val 0, matriu singular
*/
    if (piv <= 1e-15) return 0.;

/* Permutem les files i i k
  amb a[k][i] = max a[j][i], j=i,..,n-1 */
    if (k > i){
      aux = a[i];
      a[i] = a[k];
      a[k] = aux;
/* permutem files, determinant canvia de signe*/
      det = -det;
    }
```

Determinant: recurrentment (cont.)

```
      det *= a[i][i];
/* triangularitzem la matriu */
      for(k = i+1; k < n; k++){
        piv = a[k][i]/a[i][i];
        for(j = i; j < n; j++)
          a[k][j] -= a[i][j] * piv;
      }

    }
    return det;
}
```

Determinant: temps

n	Rec	Rec/n!	Def	Def/n!	Gauss
11	1.51	0.378×10^{-7}	7.97	1.99×10^{-7}	0.0000
12	18.22	0.380×10^{-7}	96.36	2.01×10^{-7}	0.0000
13	241.06	0.387×10^{-7}	1399.23	2.25×10^{-7}	0.0000
14	3337.39	0.383×10^{-7}	($\approx$20000)		0.0000