Recursão

Prof^{a}. Rose Yuri Shimizu

Algoritmos Recursivos

- São implementadas através de funções:
  - Que invocam a si mesmas
  - Chamadas de funções recursivas
- Concentra-se na solução do núcleo do problema, sendo as partes resolvidas pela recursão
- O problema é dividido em partes, que são resolvidas aplicando-se a mesma solução/fórmula
  - Repetidas aplicações da mesma solução para diversas partes
- Importante: condição de parada é necessária para terminar a recursão
- Sistemas atuais contribuem no processamento das recursividades:
  - Compiladores eficientes: otimizações contribuem para o uso eficiente dos recursos
  - Stacks: possibilitaram o empilhamento das funções

Algoritmos Recursivos - Execução

- Comportamento de uma pilha:
  - Cada iteração (função é invocada):
    - Desvia-se o fluxo de execução
    - Uma área (frame) na stack é destinada para a função: dados são empilhados, inclusive o endereço de quem chamou a função (para onde retornar - voltar para o fluxo original)
  - Última iteração:
    - Último invocado termina o seu processamento
    - É retirado da pilha e o topo da pilha retoma sua execução
- Processo de desempilhamento continua até a base da pilha
- Assim, o invocador inicial pode finalmente terminar seu processamento

Algoritmos Recursivos - Exemplo

Fatorial recursivo

$$n! = \begin{cases} 1, & \text{se } n = 0 \\ n.(n-1)!, & \text{se } n \geq 1 \end{cases}$$

```c
#include <stdio.h>

//fatorial recursivo
int fat(int n)
{
    if(n==0) return 1;
    return n * fat(n-1);
}

int main()
{
    int n = 3;

    //chamada da função e impressão do retorno
    printf("%d! = %d\n", n, fat(n));

    return 0;
}
```

```
x = fat(3)
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
      | n = 3
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |      | n = 2
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |     | n = 2
    |     | 2 * fat(1)
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |     | n = 2
    |     | 2 * fat(1)
    |     |     |n = 1
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |      | n = 2
    |      | 2 * fat(1)
    |      |      |n = 1
    |      |      |1 * fat(0)
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |     | n = 2
    |     | 2 * fat(1)
    |     |     |n = 1
    |     |     |1 * fat(0)
    |     |     |    | n = 0
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
     | n = 3
     | 3 * fat(2)
     |      | n = 2
     |      | 2 * fat(1)
     |      |      |n = 1
     |      |      |1 * fat(0)
     |      |      |     | n = 0
     |      |      |     | return 1
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |     | n = 2
    |     | 2 * fat(1)
    |     |     |n = 1
    |     |     |1 * fat(0)
    |     |     |     | n = 0
    |     |     |     | return 1
    |     |     |1 * 1
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |     | n = 2
    |     | 2 * fat(1)
    |     |     |n = 1
    |     |     |1 * fat(0)
    |     |     |     | n = 0
    |     |     |     | return 1
    |     |     |1 * 1
    |     |     |return 1
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |     | n = 2
    |     | 2 * fat(1)
    |     |     |n = 1
    |     |     |1 * fat(0)
    |     |     |    | n = 0
    |     |     |    | return 1
    |     |     |1 * 1
    |     |     |return 1
    |     | 2 * 1
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
    | n = 3
    | 3 * fat(2)
    |     | n = 2
    |     | 2 * fat(1)
    |     |     |n = 1
    |     |     |1 * fat(0)
    |     |     |    | n = 0
    |     |     |    | return 1
    |     |     |1 * 1
    |     |     |return 1
    |     | 2 * 1
    |     | return 2
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
     | n = 3
     | 3 * fat(2)
     |      | n = 2
     |      | 2 * fat(1)
     |      |      |n = 1
     |      |      |1 * fat(0)
     |      |      |     | n = 0
     |      |      |     | return 1
     |      |      |1 * 1
     |      |      |return 1
     |      | 2 * 1
     |      | return 2
     | 3 * 2
```

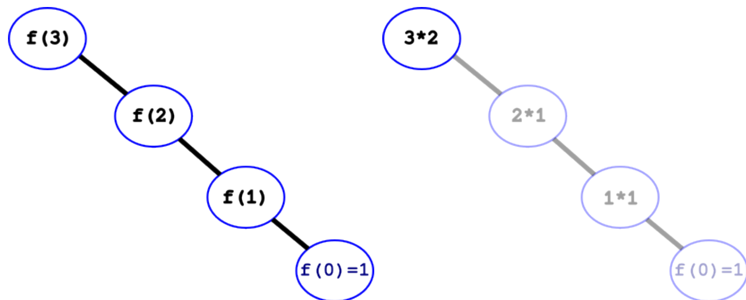
```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
     | n = 3
     | 3 * fat(2)
     |      | n = 2
     |      | 2 * fat(1)
     |      |      |n = 1
     |      |      |1 * fat(0)
     |      |      |     | n = 0
     |      |      |     | return 1
     |      |      |1 * 1
     |      |      |return 1
     |      | 2 * 1
     |      | return 2
     | 3 * 2
     | return 6
```

```c
#include <stdio.h>

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

int main()
{
    int x =  fat(3);

    return 0;
}
```

```
x = fat(3)
      | n = 3
      | 3 * fat(2)
      |       | n = 2
      |       | 2 * fat(1)
      |       |       |n = 1
      |       |       |1 * fat(0)
      |       |       |     | n = 0
      |       |       |     | return 1
      |       |       |1 * 1
      |       |       |return 1
      |       | 2 * 1
      |       | return 2
      | 3 * 2
      | return 6
x = 6
```

Representação - Árvore de recorrência



Chamadas e retornos

Algoritmos Recursivos - Exemplo

$$f(n) = \begin{cases} 0, & \text{se } n = 0 \\ 1, & \text{se } n = 1 \\ f(n-2) + f(n-1), & \text{se } n \geq 2 \end{cases}$$

```c
#include <stdio.h>
//fibonacci iterativo
int main() {
  int n = 4;
  int fn, fn1, fn2;
  fn2 = 0;  //F(0)
  fn1 = 1;  //F(1)
  fn = n;   //F(n) para n = 0 ou 1

  for(int i=2; i<=n; i++) {
    fn = fn2 + fn1; //F(n)

    //F(n-1) e F(n-2) para a próxima iteração
    fn2 = fn1; //próximo F(n-2) = atual F(n-1)
    fn1 = fn;  //próximo F(n-1) = atual F(n)
  }
  printf("%d\n", fn);
  return 0;
}
```

Algoritmos Recursivos

$$f(n) = \begin{cases} 0, & \text{se } n = 0 \\ 1, & \text{se } n = 1 \\ f(n-2) + f(n-1), & \text{se } n \geq 2 \end{cases}$$

```c
#include <stdio.h>

//fibonacci recursivo
int fib(int n){
    if(n==0) return 0;
    if(n==1) return 1;
    return fib(n-2) + fib(n-1);
}

int main(){
    int a = fib(4);
    printf("%d\n", a);
}
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
```

```c
1 int fib(int n){
2   if(n==0 || n==1) return n;
3   return fib(n-2) + fib(n-1);
4 }
5 int main(){
6   int a = fib(4);
7   printf("%d\n", a);
8 }
```

```
1 a = fib(4)
2       | n = 4
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
```

```c
1 int fib(int n){
2   if(n==0 || n==1) return n;
3   return fib(n-2) + fib(n-1);
4 }
5 int main(){
6   int a = fib(4);
7   printf("%d\n", a);
8 }
```

```
1 a = fib(4)
2     | n = 4
3     | fib(2)
4     | | fib(0)
5     | | | return 0
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
      | n = 4
      | fib(2)
      | | fib(0)
      | | | return 0
      | | fib(1)
      | | | return 1
      | | return 0 + 1
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
      | n = 4
      | fib(2)
      | | fib(0)
      | | | return 0
      | | fib(1)
      | | | return 1
      | | return 0 + 1
      | fib(3)
      | | fib(1)
      | | | return 1
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
      | n = 4
      | fib(2)
      | | fib(0)
      | | | return 0
      | | fib(1)
      | | | return 1
      | | return 0 + 1
      | fib(3)
      | | fib(1)
      | | | return 1
      | | fib(2)
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
    | | | return 1
    | | fib(2)
    | | | fib(0)
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
     | n = 4
     | fib(2)
     | | fib(0)
     | | | return 0
     | | fib(1)
     | | | return 1
     | | return 0 + 1
     | fib(3)
     | | fib(1)
     | | | return 1
     | | fib(2)
     | | | fib(0)
     | | | | return 0
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
    | | | return 1
    | | fib(2)
    | | | fib(0)
    | | | | return 0
    | | | fib(1)
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
    | | | return 1
    | | fib(2)
    | | | fib(0)
    | | | | return 0
    | | | fib(1)
    | | | | return 1
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
    | | | return 1
    | | fib(2)
    | | | fib(0)
    | | | | return 0
    | | | fib(1)
    | | | | return 1
    | | | return 0 +1
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
    | | | return 1
    | | fib(2)
    | | | fib(0)
    | | | | return 0
    | | | fib(1)
    | | | | return 1
    | | | return 0 +1
    | | return 1 + 1
```

```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
    | | | return 1
    | | fib(2)
    | | | fib(0)
    | | | | return 0
    | | | fib(1)
    | | | | return 1
    | | | return 0 +1
    | | return 1 + 1
    | return 1 + 2
```

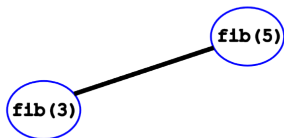
```c
int fib(int n){
  if(n==0 || n==1) return n;
  return fib(n-2) + fib(n-1);
}
int main(){
  int a = fib(4);
  printf("%d\n", a);
}
```

```
a = fib(4)
    | n = 4
    | fib(2)
    | | fib(0)
    | | | return 0
    | | fib(1)
    | | | return 1
    | | return 0 + 1
    | fib(3)
    | | fib(1)
    | | | return 1
    | | fib(2)
    | | | fib(0)
    | | | | return 0
    | | | fib(1)
    | | | | return 1
    | | | return 0 +1
    | | return 1 + 1
    | return 1 + 2
a = 3
```

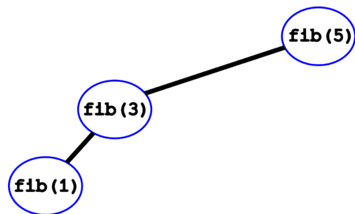
Árvore recursiva do algoritmo de Fibonacci

fib(5)

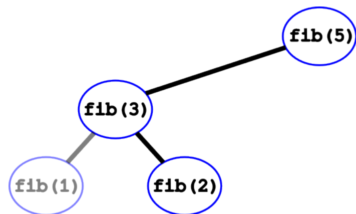
Árvore recursiva do algoritmo de Fibonacci



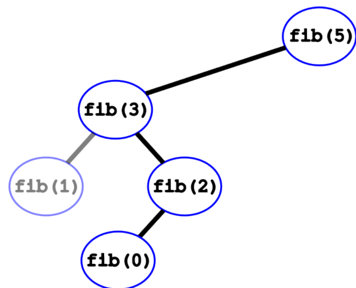
Árvore recursiva do algoritmo de Fibonacci



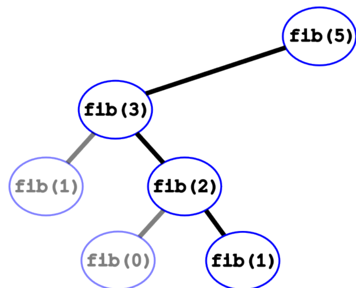
Árvore recursiva do algoritmo de Fibonacci



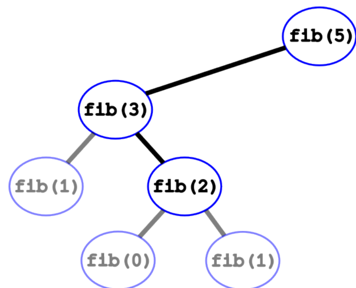
Árvore recursiva do algoritmo de Fibonacci



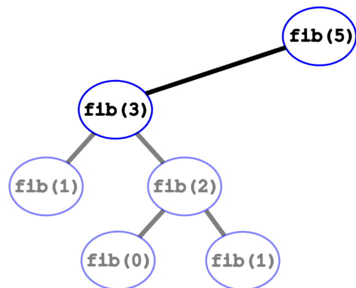
Árvore recursiva do algoritmo de Fibonacci



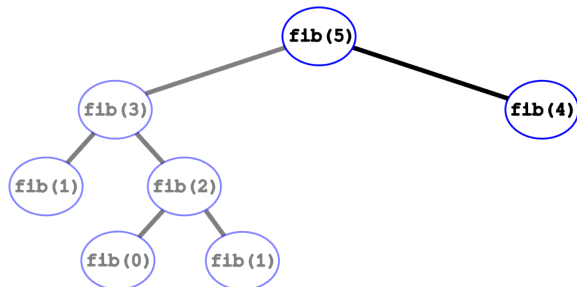
Árvore recursiva do algoritmo de Fibonacci



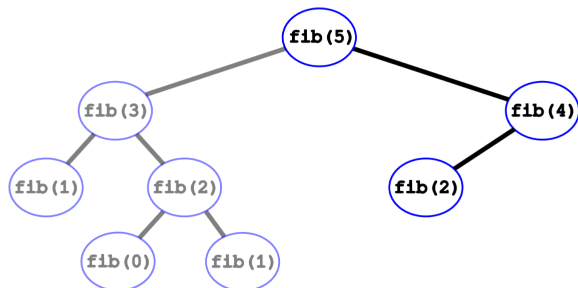
Árvore recursiva do algoritmo de Fibonacci



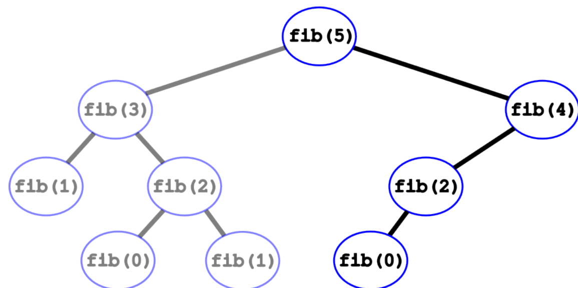
Árvore recursiva do algoritmo de Fibonacci



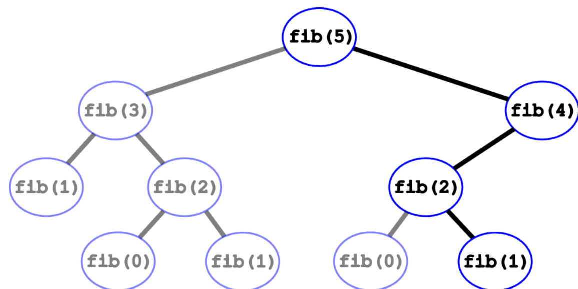
Árvore recursiva do algoritmo de Fibonacci



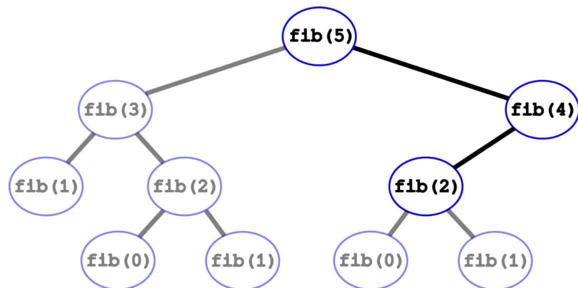
Árvore recursiva do algoritmo de Fibonacci



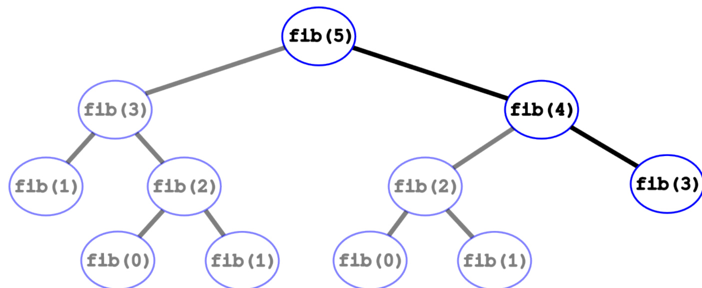
Árvore recursiva do algoritmo de Fibonacci



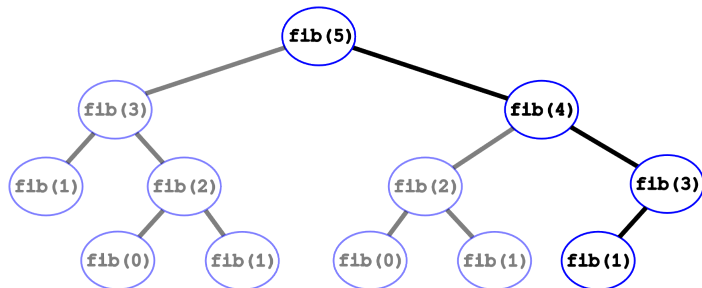
Árvore recursiva do algoritmo de Fibonacci



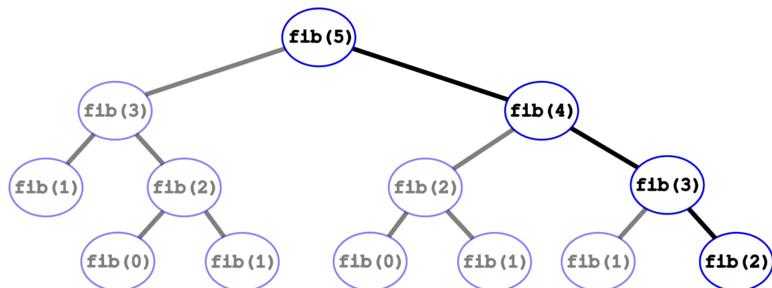
Árvore recursiva do algoritmo de Fibonacci



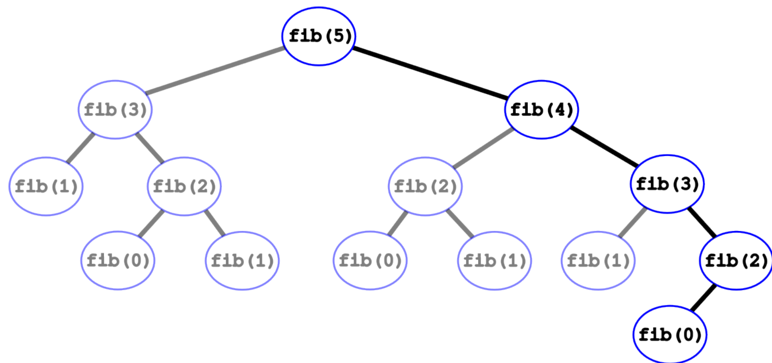
Árvore recursiva do algoritmo de Fibonacci



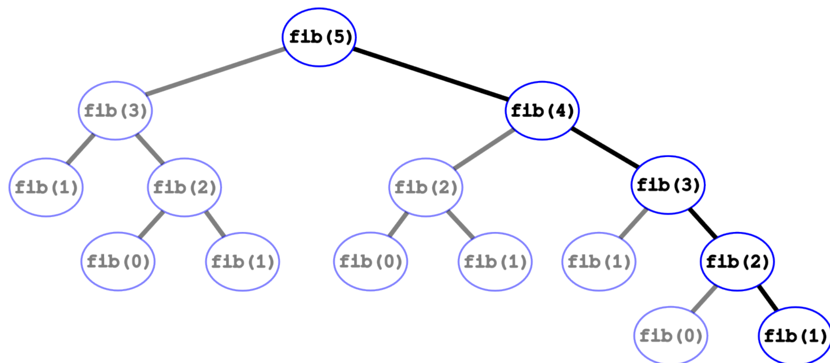
Árvore recursiva do algoritmo de Fibonacci



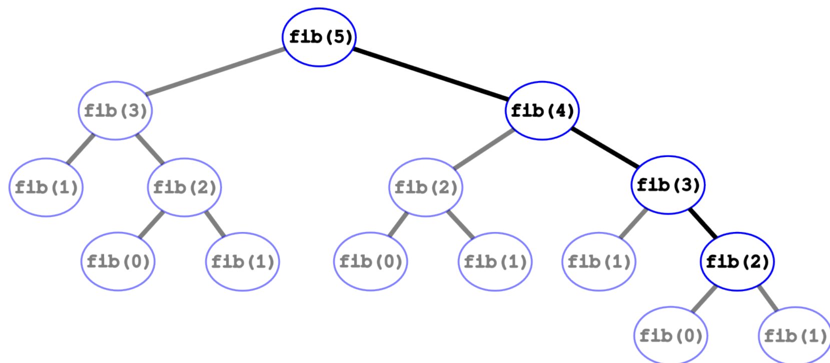
Árvore recursiva do algoritmo de Fibonacci



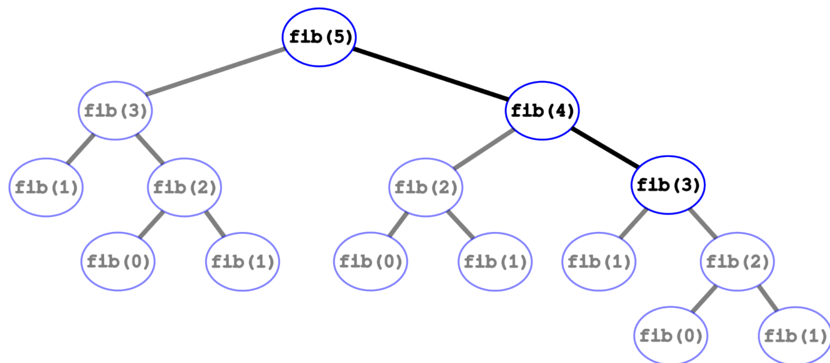
Árvore recursiva do algoritmo de Fibonacci



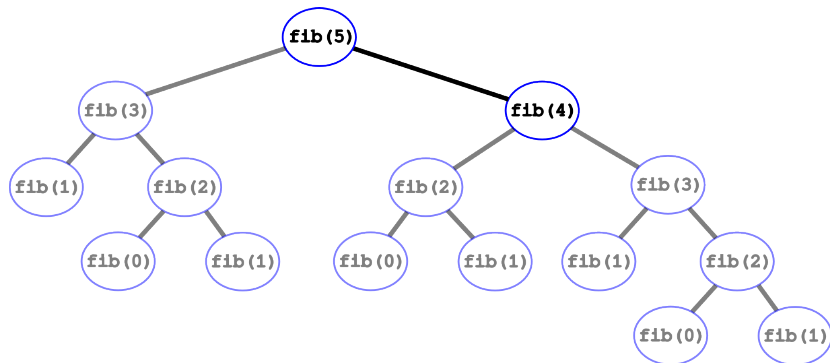
Árvore recursiva do algoritmo de Fibonacci



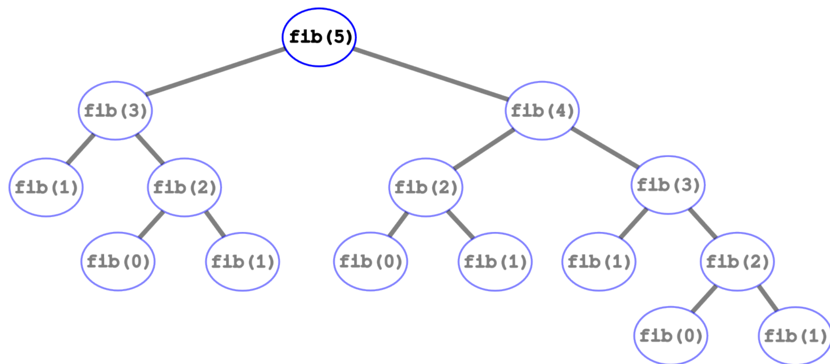
Árvore recursiva do algoritmo de Fibonacci



Árvore recursiva do algoritmo de Fibonacci



Árvore recursiva do algoritmo de Fibonacci



$+$ eficiente: iterativo ou recursivo?

Algoritmos Recursivos

```
1 fibonacci(n)
2 inicio
3   se (n==0 || n==1) retorne n;
4
5   se ainda nao foi calculado
6   inicio
7     calcule e guarde
8   fim
9
10  retorne o calculado para n
11 fim
```

- Alternativa: utilizar a técnica da memoização
  - Armazenamento de resultados de chamadas de função custosas
  - Lista com resultados
  - Só calcular (chamada recursiva) se ainda não foi calculado
- Vetor (ex. v[n]):
  - n: índice corresponde ao número do qual está calculando-se o fibonacci
  - v[n]: conteúdo, corresponde ao fibonacci de n

Algoritmos Recursivos

- Pode ser aplicado em problemas de:
  - Planejamento de caminhos em robótica
  - Problemas de tentativa e erro (*backtracking*: voltar e explorar outra solução)
  - Compiladores (analisadores léxicos)
  - **Manipulação das estrutura de dados** (formas de armazenamento de dados)
  - **Algoritmos de pesquisas, ordenação**

Algoritmos Recursivos - Exemplo

- Resolver expressões na notação polonesa: sem necessidade de utilizar parênteses
- Notação pré-fixa
- Operador antes dos operandos: $+ab = a + b$
- Lógica: se for operador, busca até achar operando, busca até achar operando e resolve
- `* + 7 * * 4 6 + 8 9 5`

```
1 busca(expr)
2 inicio
3   se expr == operando
4     retorna operando
5
6   se expr == operador
7     busca(expr) operador busca(expr)
8 fim
```

```c
//Exemplo do livro do Sedgewick
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;
```

```
1 //Exemplo do livro do Sedgewick
2 char *a = "* + 7 * * 4 6 + 8 9 5";
3 int i=0;
4
5 int eval() {
6     int x=0;
7     while(a[i] == ' ') i++; //avança enquanto for espaço
```

```
1 //Exemplo do livro do Sedgewick
2 char *a = "* + 7 * * 4 6 + 8 9 5";
3 int i=0;
4
5 int eval() {
6     int x=0;
7     while(a[i] == ' ') i++; //avança enquanto for espaço
8     if(a[i] == '+') {
9         i++;
10        return eval() + eval();
11    }
```

```
1 //Exemplo do livro do Sedgewick
2 char *a = "* + 7 * * 4 6 + 8 9 5";
3 int i=0;
4
5 int eval() {
6     int x=0;
7     while(a[i] == ' ') i++; //avança enquanto for espaço
8     if(a[i] == '+') {
9         i++;
10        return eval() + eval();
11    }
12    if(a[i] == '*') {
13        i++;
14        return eval() * eval();
15    }
16
```

```c
//Exemplo do livro do Sedgewick
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
    int x=0;
    while(a[i] == ' ') i++; //avança enquanto for espaço
    if(a[i] == '+') {
        i++;
        return eval() + eval();
    }
    if(a[i] == '*') {
        i++;
        return eval() * eval();
    }

    //enquanto for número
    //converte um dígito(char) em número(int)
    while((a[i] >= '0') && (a[i] <= '9'))
        x = 10*x + (a[i++]-'0'); //tabela ascii

    //"91": '9' e '1'
    //  10*0 + ('9'-'0') =  0 + (57-48) =  9
    //  10*9 + ('1'-'0') = 90 + (49-48) = 91

    return x;
}
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

$* + 7 * * 4\ 6 + 8\ 9\ 5 = (7+((4*6)*(8+9)))*5$

```
eval() * + 7 * * 4 6 + 8 9 5
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

$* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5$

```
eval() * + 7 * * 4 6 + 8 9 5
|   eval() + 7 * * 4 6 + 8 9
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
eval() * + 7 * * 4 6 + 8 9 5
|  eval() + 7 * * 4 6 + 8 9
|  |  eval() return 7
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
eval() * + 7 * * 4 6 + 8 9 5
|   eval() + 7 * * 4 6 + 8 9
|   |   eval() return 7
|   |   eval() * * 4 6 + 8 9
```

```
1  char *a = "* + 7 * * 4 6 + 8 9 5";
2  int i=0;
3
4  int eval() {
5    int x=0;
6    while(a[i] == ' ') i++;
7    if(a[i] == '+') {
8      i++;
9      return eval() + eval();
10   }
11   if(a[i] == '*') {
12     i++;
13     return eval() * eval();
14   }
15
16   while(a[i]>='0' && a[i]<='9')
17     x = 10*x + (a[i++]-'0');
18
19   return x;
20 }
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
1 eval() * + 7 * * 4 6 + 8 9 5
2 |   eval() + 7 * * 4 6 + 8 9
3 |   |   eval() return 7
4 |   |   eval() * * 4 6 + 8 9
5 |   |   |   eval() * 4 6
```

```
1  char *a = "* + 7 * * 4 6 + 8 9 5";
2  int i=0;
3
4  int eval() {
5    int x=0;
6    while(a[i] == ' ') i++;
7    if(a[i] == '+') {
8      i++;
9      return eval() + eval();
10   }
11   if(a[i] == '*') {
12     i++;
13     return eval() * eval();
14   }
15
16   while(a[i]>='0' && a[i]<='9')
17     x = 10*x + (a[i++]-'0');
18
19   return x;
20 }
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
1 eval() * + 7 * * 4 6 + 8 9 5
2 |   eval() + 7 * * 4 6 + 8 9
3 |   |   eval() return 7
4 |   |   eval() * * 4 6 + 8 9
5 |   |   |   eval() * 4 6
6 |   |   |   |   eval() return 4
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
eval() * + 7 * * 4 6 + 8 9 5
|  eval() + 7 * * 4 6 + 8 9
|  |  eval() return 7
|  |  eval() * * 4 6 + 8 9
|  |  |  eval() * 4 6
|  |  |  |  eval() return 4
|  |  |  |  eval() return 6
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

$* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5$

```
eval() * + 7 * * 4 6 + 8 9 5
|  eval() + 7 * * 4 6 + 8 9
|  |  eval() return 7
|  |  eval() * * 4 6 + 8 9
|  |  |  eval() * 4 6
|  |  |  |  eval() return 4
|  |  |  |  eval() return 6
|  |  |  |_ return  4 * 6 = 24
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
eval() * + 7 * * 4 6 + 8 9 5
|   eval() + 7 * * 4 6 + 8 9
|   |   eval() return 7
|   |   eval() * * 4 6 + 8 9
|   |   |   eval() * 4 6
|   |   |   |   eval() return 4
|   |   |   |   eval() return 6
|   |   |   |_ return  4 * 6 = 24
|   |   |
|   |   |   eval() + 8 9
```

```
1  char *a = "* + 7 * * 4 6 + 8 9 5";
2  int i=0;
3
4  int eval() {
5    int x=0;
6    while(a[i] == ' ') i++;
7    if(a[i] == '+') {
8      i++;
9      return eval() + eval();
10   }
11   if(a[i] == '*') {
12     i++;
13     return eval() * eval();
14   }
15
16   while(a[i]>='0' && a[i]<='9')
17     x = 10*x + (a[i++]-'0');
18
19   return x;
20 }
```

```
* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5
```

```
1  eval() * + 7 * * 4 6 + 8 9 5
2  |   eval() + 7 * * 4 6 + 8 9
3  |   |   eval() return 7
4  |   |   eval() * * 4 6 + 8 9
5  |   |   |   eval() * 4 6
6  |   |   |   |   eval() return 4
7  |   |   |   |   eval() return 6
8  |   |   |   |_  return  4 * 6 = 24
9  |   |   |
10 |   |   |   eval() + 8 9
11 |   |   |   |   eval() return 8
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

$* + 7 * * 4\ 6 + 8\ 9\ 5 = (7+((4*6)*(8+9)))*5$

```
eval() * + 7 * * 4 6 + 8 9 5
|  eval() + 7 * * 4 6 + 8 9
|  |  eval() return 7
|  |  eval() * * 4 6 + 8 9
|  |  |  eval() * 4 6
|  |  |  |  eval() return 4
|  |  |  |  eval() return 6
|  |  |  |_ return  4 * 6 = 24
|  |  |
|  |  |  eval() + 8 9
|  |  |  |  eval() return 8
|  |  |  |  eval() return 9
```

```
1 char *a = "* + 7 * * 4 6 + 8 9 5";
2 int i=0;
3
4 int eval() {
5   int x=0;
6   while(a[i] == ' ') i++;
7   if(a[i] == '+') {
8     i++;
9     return eval() + eval();
10  }
11  if(a[i] == '*') {
12    i++;
13    return eval() * eval();
14  }
15
16  while(a[i]>='0' && a[i]<='9')
17    x = 10*x + (a[i++]-'0');
18
19  return x;
20 }
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
1  eval() * + 7 * * 4 6 + 8 9 5
2  |  eval() + 7 * * 4 6 + 8 9
3  |  |  eval() return 7
4  |  |  eval() * * 4 6 + 8 9
5  |  |  |  eval() * 4 6
6  |  |  |  |  eval() return 4
7  |  |  |  |  eval() return 6
8  |  |  |  |_ return  4 * 6 = 24
9  |  |  |
10 |  |  |  eval() + 8 9
11 |  |  |  |  eval() return 8
12 |  |  |  |  eval() return 9
13 |  |  |  |_ return 8 + 9 = 17
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
eval() * + 7 * * 4 6 + 8 9 5
|  eval() + 7 * * 4 6 + 8 9
|  |  eval() return 7
|  |  eval() * * 4 6 + 8 9
|  |  |  eval() * 4 6
|  |  |  |  eval() return 4
|  |  |  |  eval() return 6
|  |  |  |_ return  4 * 6 = 24
|  |  |
|  |  |  eval() + 8 9
|  |  |  |  eval() return 8
|  |  |  |  eval() return 9
|  |  |  |_ return 8 + 9 = 17
|  |  |_ return 24 * 17 = 408
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

$* + 7 * * 4\ 6 + 8\ 9\ 5 = (7+((4*6)*(8+9)))*5$

```
eval() * + 7 * * 4 6 + 8 9 5
|   eval() + 7 * * 4 6 + 8 9
|   |   eval() return 7
|   |   eval() * * 4 6 + 8 9
|   |   |   eval() * 4 6
|   |   |   |   eval() return 4
|   |   |   |   eval() return 6
|   |   |   |_ return  4 * 6 = 24
|   |   |
|   |   |   eval() + 8 9
|   |   |   |   eval() return 8
|   |   |   |   eval() return 9
|   |   |   |_ return 8 + 9 = 17
|   |   |_ return 24 * 17 = 408
|   |_ return 7 + 408 = 415
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

```
* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5
```

```
eval() * + 7 * * 4 6 + 8 9 5
|   eval() + 7 * * 4 6 + 8 9
|   |   eval() return 7
|   |   eval() * * 4 6 + 8 9
|   |   |   eval() * 4 6
|   |   |   |   eval() return 4
|   |   |   |   eval() return 6
|   |   |   |_  return  4 * 6 = 24
|   |   |
|   |   |   eval() + 8 9
|   |   |   |   eval() return 8
|   |   |   |   eval() return 9
|   |   |   |_  return 8 + 9 = 17
|   |   |_  return 24 * 17 = 408
|   |_  return 7 + 408 = 415
|   eval() return 5
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

$* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5$

```
eval() * + 7 * * 4 6 + 8 9 5
|  eval() + 7 * * 4 6 + 8 9
|  |  eval() return 7
|  |  eval() * * 4 6 + 8 9
|  |  |  eval() * 4 6
|  |  |  |  eval() return 4
|  |  |  |  eval() return 6
|  |  |  |_ return  4 * 6 = 24
|  |  |
|  |  |  eval() + 8 9
|  |  |  |  eval() return 8
|  |  |  |  eval() return 9
|  |  |  |_ return 8 + 9 = 17
|  |  |_ return 24 * 17 = 408
|  |_ return 7 + 408 = 415
|  eval() return 5
|_ return 5*415 = 2075
```

```c
char *a = "* + 7 * * 4 6 + 8 9 5";
int i=0;

int eval() {
  int x=0;
  while(a[i] == ' ') i++;
  if(a[i] == '+') {
    i++;
    return eval() + eval();
  }
  if(a[i] == '*') {
    i++;
    return eval() * eval();
  }

  while(a[i]>='0' && a[i]<='9')
    x = 10*x + (a[i++]-'0');

  return x;
}
```

* + 7 * * 4 6 + 8 9 5 = (7+((4*6)*(8+9)))*5

```
eval() * + 7 * * 4 6 + 8 9 5
|   eval() + 7 * * 4 6 + 8 9
|   |   eval() return 7
|   |   eval() * * 4 6 + 8 9
|   |   |   eval() * 4 6
|   |   |   |   eval() return 4
|   |   |   |   eval() return 6
|   |   |   |_ return  4 * 6 = 24
|   |   |
|   |   |   eval() + 8 9
|   |   |   |   eval() return 8
|   |   |   |   eval() return 9
|   |   |   |_ return 8 + 9 = 17
|   |   |_ return 24 * 17 = 408
|   |_ return 7 + 408 = 415
|   eval() return 5
|_ return 5*415 = 2075
```

Algoritmos recursivos - Exemplo

```
1 int max(int n, int v[]) {
2   if (n == 1) return v[0];
3   else {
4     int x = max(n-1, v);
5
6     if (x > v[n-1]) return x;
7     else return v[n-1];
8   }
9 }
```

```
1 v[3] = {77, 88, 66}
2 max(3, v)
3 | max(2, v)
4 | | max(1, v)
5 | | |_ return 77
6 | |_ return 88
7 |_ return 88
```

Algoritmos recursivos - Exemplo

<table>
<tr><td>-</td><td>-</td><td>-</td><td>1</td><td>2</td><td>1</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>-</td><td>-</td><td>1</td><td>1</td><td></td><td>1</td><td>1</td><td>-</td><td>-</td></tr>
<tr><td>-</td><td>-</td><td>1</td><td></td><td></td><td></td><td>3</td><td>-</td><td>-</td></tr>
<tr><td>-</td><td>-</td><td>1</td><td></td><td></td><td></td><td>2</td><td>-</td><td>-</td></tr>
<tr><td>-</td><td>2</td><td>1</td><td></td><td></td><td></td><td>1</td><td>-</td><td>-</td></tr>
<tr><td>-</td><td>1</td><td></td><td></td><td></td><td></td><td>1</td><td>-</td><td>-</td></tr>
<tr><td>-</td><td>1</td><td></td><td>[l,c]</td><td></td><td></td><td>1</td><td>2</td><td>-</td></tr>
<tr><td>-</td><td>1</td><td></td><td></td><td></td><td></td><td></td><td>1</td><td>-</td></tr>
<tr><td>-</td><td>2</td><td>1</td><td></td><td></td><td></td><td></td><td>1</td><td>-</td></tr>
<tr><td>-</td><td>-</td><td>3</td><td>1</td><td></td><td>2</td><td>1</td><td>1</td><td>-</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>1</td><td>1</td><td>1</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table>

- Campo minado - abrir casas
- A partir da posição `l` e `c`, como abrir os adjacentes-

```
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -   [l,c]   -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
-     -     -     -     -     -     -     -     -
```

- Campo minado - abrir casas
- A partir da posição `l` e `c`, como abrir os adjacentes?

```
[-l,-c][-l, c][-l,+c]
[ l,-c][ l, c][ l,+c]
[+l,-c][+l, c][+l,+c]
```

```
1  for(int i=l-1; i<=l+1; i++) {
```

```
–     –     –       –       –       –     –     –     –
–     –     –       –       –       –     –     –     –
–     –     –       –       –       –     –     –     –
–     –     –       –       –       –     –     –     –
–     –     –       –       –       –     –     –     –
–     –  [-l,-c] [-l, c] [-l,+c]    –     –     –     –
–     –  [ l,-c] [ l, c] [ l,+c]    –     –     –     –
–     –  [+l,-c] [+l, c] [+l,+c]    –     –     –     –
–     –     –       –       –       –     –     –     –
–     –     –       –       –       –     –     –     –
–     –     –       –       –       –     –     –     –
–     –     –       –       –       –     –     –     –
```

```
1  for(int i=l-1; i<=l+1; i++) {
2    for(int j=c-1; j<=c+1; j++) {
```

```
-  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -
-  -  [-l,-c][-l, c][-l,+c]  -  -  -  -
-  -  [ l,-c][ l, c][ l,+c]  -  -  -  -
-  -  [+l,-c][+l, c][+l,+c]  -  -  -  -
-  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -
-  -  -  -  -  -  -  -  -
```

```
1  for(int i=l-1; i<=l+1; i++) {
2    for(int j=c-1; j<=c+1; j++) {
3        if(i>=0 && i<M && j>=0 && j<N) //limites
```

```
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  [-l,-c][-l, c][-l,+c]  –  –  –  –
–  –  [ l,-c][ l, c][ l,+c]  –  –  –  –
–  –  [+l,-c][+l, c][+l,+c]  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
```

```
1  for(int i=l-1; i<=l+1; i++) {
2    for(int j=c-1; j<=c+1; j++) {
3        if(i>=0 && i<M && j>=0 && j<N) //limites
4          if(campo[i][j].visivel==0) //não aberta
```

```
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  [-l,-c][-l, c][-l,+c]  –  –  –  –
–  –  [ l,-c][ l, c][ l,+c]  –  –  –  –
–  –  [+l,-c][+l, c][+l,+c]  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
–  –  –  –  –  –  –  –  –
```

```
1  for(int i=l-1; i<=l+1; i++) {
2    for(int j=c-1; j<=c+1; j++) {
3        if(i>=0 && i<M && j>=0 && j<N) //limites
4          if(campo[i][j].visivel==0) //não aberta
5            campo[i][j].visivel = 1; //abra
6    }
7  }
```

```
-    -    -       -       -       -    -    -    -
-    -    -       -       -       -    -    -    -
-    -    -       -       -       -    -    -    -
-    -    -       -       -       -    -    -    -
-    ?    ?       ?       ?       ?    -    -    -
-    ?  [-l,-c][-l, c][-l,+c]    ?    -    -    -
-    ?  [ l,-c][ l, c][ l,+c]    ?    -    -    -
-    ?  [+l,-c][+l, c][+l,+c]    ?    -    -    -
-    ?    ?       ?       ?       ?    -    -    -
-    -    -       -       -       -    -    -    -
-    -    -       -       -       -    -    -    -
-    -    -       -       -       -    -    -    -
```

- Campo minado - abrir casas
- A partir da posição `l` e `c`, como abrir os adjacentes?
- E os adjacentes dos adjacentes?

```
–    –       –       –       –       –    –    –    –
–    –       –       –       –       –    –    –    –
–    –       –       –       –       –    –    –    –
–    –       –       –       –       –    –    –    –
–    ?       ?       ?       ?       ?    –    –    –
–    ?    [-l,-c][-l, c][-l,+c]      ?    –    –    –
–    ?    [ l,-c][ l, c][ l,+c]      ?    –    –    –
–    ?    [+l,-c][+l, c][+l,+c]      ?    –    –    –
–    ?       ?       ?       ?       ?    –    –    –
–    –       –       –       –       –    –    –    –
–    –       –       –       –       –    –    –    –
–    –       –       –       –       –    –    –    –
```

- Campo minado - abrir casas
- A partir da posição `l` e `c`, como abrir os adjacentes?
- E os adjacentes dos adjacentes? Recursão

```c
#define M 20
#define N 20
int abrir_mapa(struct area campo[M][N], int l, int c) {
  campo[l][c].visivel = 1;

  if(campo[l][c].item!=0)
    return campo[l][c].item;

  for(int i=l-1; i<=l+1; i++) {
    for(int j=c-1; j<=c+1; j++) {
      if(i>=0 && i<M && j>=0 && j<N &&
         campo[i][j].visivel==0) {
           abrir_mapa(m, n, campo, i, j);
      }
    }
  }

  return 0;
}
```

Algoritmos recursivos - Exemplo

```
1 void subsets(int sub[], int v[], int n, int i, int fim) {
2   for(int j = 0; j < fim; j++)  printf("%d ", sub[j]);
3   printf("\n");
4 
5   for(; i < n; i++) {
6     //inclui o elemento
7     sub[fim] = v[i];
8 
9     //gera o subconjunto com o elemento incluído
10    subsets(sub, v, n, i+1, fim+1);
11
12    //‘fim’ não é atualizado,
13    //permancendo o valor anterior, ou seja,
14    //excluíndo o elemento que incluído nesta iteração
15  }
16 }
17 int main() {
18     int v[] = { 1, 2, 3, 4 };
19     int sub[4];
20     subsets(sub, v, 4, 0, 0);
21     return 0;
22 }
```

```
1
1 2
1 2 3
1 2 3 4
1 2 4
1 3
1 3 4
1 4
2
2 3
2 3 4
2 4
3
3 4
4
```

- Backtracking: técnica algorítmica de resolução de problemas que envolve encontrar uma solução de forma incremental, tentando diferentes opções e desfazendo-as se elas levarem a um beco sem saída.
- Procurar solução, se não achar volta e busca por outro caminho: subconjunto, labirinto

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1 subsets(sub, v, 3, 0, 0)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1 subsets(sub, v, 3, 0, 0)
2 |   \n
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1 subsets(sub, v, 3, 0, 0)
2 |   \n
3 |   sub[0] = v[0]
4 |   subsets(sub, v, 3, 0+1, 0+1)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1 subsets(sub, v, 3, 0, 0)
2 |   \n
3 |   sub[0] = v[0]
4 |   subsets(sub, v, 3, 0+1, 0+1)
5 |   |   1
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1 subsets(sub, v, 3, 0, 0)
2 |   \n
3 |   sub[0] = v[0]
4 |   subsets(sub, v, 3, 0+1, 0+1)
5 |   |   1
6 |   |   sub[1] = v[1]
7 |   |   subsets(sub, v, 3, 1+1, 1+1)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1 subsets(sub, v, 3, 0, 0)
2 |   \n
3 |   sub[0] = v[0]
4 |   subsets(sub, v, 3, 0+1, 0+1)
5 |   |   1
6 |   |   sub[1] = v[1]
7 |   |   subsets(sub, v, 3, 1+1, 1+1)
8 |   |   |   1 2
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
12 |   |   sub[1] = v[2]
13 |   |   subsets(sub, v, 3, 2+1, 1+1)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
12 |   |   sub[1] = v[2]
13 |   |   subsets(sub, v, 3, 2+1, 1+1)
14 |   |   |   1 3
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
12 |   |   sub[1] = v[2]
13 |   |   subsets(sub, v, 3, 2+1, 1+1)
14 |   |   |   1 3
15 |   sub[0] = v[1]
16 |   subsets(sub, v, 3, 1+1, 0+1)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
12 |   |   sub[1] = v[2]
13 |   |   subsets(sub, v, 3, 2+1, 1+1)
14 |   |   |   1 3
15 |   sub[0] = v[1]
16 |   subsets(sub, v, 3, 1+1, 0+1)
17 |   |   2
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
12 |   |   sub[1] = v[2]
13 |   |   subsets(sub, v, 3, 2+1, 1+1)
14 |   |   |   1 3
15 |   sub[0] = v[1]
16 |   subsets(sub, v, 3, 1+1, 0+1)
17 |   |   2
18 |   |   sub[1] = v[2]
19 |   |   subsets(sub, v, 3, 2+1, 1+1)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |  \n
3  |  sub[0] = v[0]
4  |  subsets(sub, v, 3, 0+1, 0+1)
5  |  |  1
6  |  |  sub[1] = v[1]
7  |  |  subsets(sub, v, 3, 1+1, 1+1)
8  |  |  |  1 2
9  |  |  |  sub[2] = v[2]
10 |  |  |  subsets(sub, v, 3, 2+1, 2+1)
11 |  |  |  |  1 2 3
12 |  |  sub[1] = v[2]
13 |  |  subsets(sub, v, 3, 2+1, 1+1)
14 |  |  |  1 3
15 |  sub[0] = v[1]
16 |  subsets(sub, v, 3, 1+1, 0+1)
17 |  |  2
18 |  |  sub[1] = v[2]
19 |  |  subsets(sub, v, 3, 2+1, 1+1)
20 |  |  |  2 3
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
12 |   |   sub[1] = v[2]
13 |   |   subsets(sub, v, 3, 2+1, 1+1)
14 |   |   |   1 3
15 |   sub[0] = v[1]
16 |   subsets(sub, v, 3, 1+1, 0+1)
17 |   |   2
18 |   |   sub[1] = v[2]
19 |   |   subsets(sub, v, 3, 2+1, 1+1)
20 |   |   |   2 3
21 |   sub[0] = v[2]
22 |   subsets(sub, v, 3, 2+1, 0+1)
```

```c
void subsets(int sub[],
             int v[], int n,
             int i, int fim) {

  for(int j=0; j<fim; j++)
    printf("%d ", sub[j]);
  printf("\n");

  for(; i < n; i++) {
    sub[fim] = v[i];
    subsets(sub, v, n, i+1, fim+1);
  }
}

int main() {
    int v[] = { 1, 2, 3 };
    int sub[3];
    subsets(sub, v, 3, 0, 0);

    return 0;
}
```

```
1  subsets(sub, v, 3, 0, 0)
2  |   \n
3  |   sub[0] = v[0]
4  |   subsets(sub, v, 3, 0+1, 0+1)
5  |   |   1
6  |   |   sub[1] = v[1]
7  |   |   subsets(sub, v, 3, 1+1, 1+1)
8  |   |   |   1 2
9  |   |   |   sub[2] = v[2]
10 |   |   |   subsets(sub, v, 3, 2+1, 2+1)
11 |   |   |   |   1 2 3
12 |   |   sub[1] = v[2]
13 |   |   subsets(sub, v, 3, 2+1, 1+1)
14 |   |   |   1 3
15 |   sub[0] = v[1]
16 |   subsets(sub, v, 3, 1+1, 0+1)
17 |   |   2
18 |   |   sub[1] = v[2]
19 |   |   subsets(sub, v, 3, 2+1, 1+1)
20 |   |   |   2 3
21 |   sub[0] = v[2]
22 |   subsets(sub, v, 3, 2+1, 0+1)
23 |   |   3
```

Algoritmos recursivos - Exemplo

```c
void anagram(char str[], int k) {
  char tmp;
  int i, len = strlen(str);
  if(k == len) printf("%s\n", str);
  else
    for(i = k; i < len; i++) {
        swap_char(str, k, i); //1 troca
        anagram(str, k + 1);  //próximas permutas
        swap_char(str, i, k); //volta ao original
    }
}
```

anagram("abc", 0)

```
swap("abc", 0, 0)
anagram("abc", 1)
  swap("abc", 1, 1)
  anagram("abc", 2)
    swap("abc", 2, 2)
    anagram("abc", 3)
      "abc"
    swap("abc", 2, 2)
  swap("abc", 1, 1)

  swap("abc", 1, 2)
  anagram("acb", 2)
    swap("acb", 2, 2)
    anagram("acb", 3)
      "acb"
    swap("acb", 2, 2)
  swap("acb", 2, 1)
swap("abc", 0, 0)

```

```
swap("abc", 0, 1)
anagram("bac", 1)
  swap("bac", 1, 1)
  anagram("bac", 2)
    swap("bac", 2, 2)
    anagram("bac", 3)
      "bac"
    swap("bac", 2, 2)
  swap("bac", 1, 1)

  swap("bac", 1, 2)
  anagram("bca", 2)
    swap("bca", 2, 2)
    anagram("bca", 3)
      "bca"
    swap("bca", 2, 2)
  swap("bca", 2, 1)
swap("bac", 1, 0)

```

```
swap("abc", 0, 2)
anagram("cba", 1)
  swap("cba", 1, 1)
  anagram("cba", 2)
    swap("cba", 2, 2)
    anagram("cba", 3)
      "cba"
    swap("cba", 2, 2)
  swap("cba", 1, 1)

  swap("cba", 1, 2)
  anagram("cab", 2)
    swap("cab", 2, 2)
    anagram("cab", 3)
      "cab"
    swap("cab", 2, 2)
  swap("cab", 2, 1)
swap("cba", 2, 0)
```

Algoritmos recursivos - Exemplo

```
1  int expr_check(char *s, int t, int *p) {
2    int i;
3    for(i=t+1; s[i]!='\0'; i++){
4      if(*p>0 && (s[i]=='}' && s[t]=='{') ||
5                 (s[i]==')' && s[t]=='(') ||
6                 (s[i]==']' && s[t]=='[') ) {
7        *p=*p-1;
8        return i;
9      }
10
11     *p = *p+1;
12     i = expr_check(s, i, p);
13   }
14   return i;
15 }
16 int expr(char *s, int a) {
17   int p=0;
18   for(int i=0; i>=0 && s[i]!='\0'; i++){
19     p++;
20     i = expr_check(s, i, &p);
21   }
22   if(p>0) printf("N\n");
23   else printf("S\n");
24 }
```

- Análise léxica: `[(){[()][]}]`
- Abre, busca até fechar
- Verifica na volta se abre e fecha

Algoritmos recursivos - Exemplo

```c
/*    0 1 2 3
    {2,1,3,0}

      0 1 2 3
    {3,1,0,2}
*/
void inverte(int i, int v[], int tam) {
  if(i<tam){
    int j = v[i];
    inverte(i+1, v, tam);
    v[j] = i;

    for(int p=0; p<tam; p++)
      printf("%d ", v[p]);
    printf("\n");
  }
}
```

```
inverte(0, v, 4)
|   j = v[0] //2
|   inverte(0+1, v, 4)
```

Algoritmos recursivos - Exemplo

```c
/*    0 1 2 3
    {2,1,3,0}

      0 1 2 3
    {3,1,0,2}
*/
void inverte(int i, int v[], int tam) {
  if(i<tam){
    int j = v[i];
    inverte(i+1, v, tam);
    v[j] = i;

    for(int p=0; p<tam; p++)
      printf("%d ", v[p]);
    printf("\n");
  }
}
```

```
inverte(0, v, 4)
|   j = v[0] //2
|   inverte(0+1, v, 4)
|   |   j = v[1] //1
|   |   inverte(1+1, v, 4)
```

Algoritmos recursivos - Exemplo

```
1  /*    0 1 2 3
2      {2,1,3,0}
3
4        0 1 2 3
5      {3,1,0,2}
6  */
7  void inverte(int i, int v[], int tam) {
8    if(i<tam){
9      int j = v[i];
10     inverte(i+1, v, tam);
11     v[j] = i;
12
13     for(int p=0; p<tam; p++)
14       printf("%d ", v[p]);
15     printf("\n");
16   }
17 }
```

```
1 inverte(0, v, 4)
2 |   j = v[0] //2
3 |   inverte(0+1, v, 4)
4 |   |   j = v[1] //1
5 |   |   inverte(1+1, v, 4)
6 |   |   |   j = v[2] //3
7 |   |   |   inverte(2+1, v, 4)
```

Algoritmos recursivos - Exemplo

```c
/*    0 1 2 3
    {2,1,3,0}

     0 1 2 3
    {3,1,0,2}
*/
void inverte(int i, int v[], int tam) {
  if(i<tam){
    int j = v[i];
    inverte(i+1, v, tam);
    v[j] = i;

    for(int p=0; p<tam; p++)
      printf("%d ", v[p]);
    printf("\n");
  }
}
```

```
inverte(0, v, 4)
|   j = v[0] //2
|   inverte(0+1, v, 4)
|   |   j = v[1] //1
|   |   inverte(1+1, v, 4)
|   |   |   j = v[2] //3
|   |   |   inverte(2+1, v, 4)
|   |   |   |   j = v[3] //0
|   |   |   |   inverte(3+1, v, 4)
```

Algoritmos recursivos - Exemplo

```c
/*    0 1 2 3
    {2,1,3,0}

      0 1 2 3
    {3,1,0,2}
*/
void inverte(int i, int v[], int tam) {
  if(i<tam){
    int j = v[i];
    inverte(i+1, v, tam);
    v[j] = i;

    for(int p=0; p<tam; p++)
      printf("%d ", v[p]);
    printf("\n");
  }
}
```

```
inverte(0, v, 4)
|   j = v[0] //2
|   inverte(0+1, v, 4)
|   |   j = v[1] //1
|   |   inverte(1+1, v, 4)
|   |   |   j = v[2] //3
|   |   |   inverte(2+1, v, 4)
|   |   |   |   j = v[3] //0
|   |   |   |   inverte(3+1, v, 4)
|   |   |   |   v[0] = 3
|   |   |   |   3,1,3,0
```

Algoritmos recursivos - Exemplo

```c
/*    0 1 2 3
    {2,1,3,0}

      0 1 2 3
    {3,1,0,2}
*/
void inverte(int i, int v[], int tam) {
  if(i<tam){
    int j = v[i];
    inverte(i+1, v, tam);
    v[j] = i;

    for(int p=0; p<tam; p++)
      printf("%d ", v[p]);
    printf("\n");
  }
}
```

```
inverte(0, v, 4)
|   j = v[0] //2
|   inverte(0+1, v, 4)
|   |   j = v[1] //1
|   |   inverte(1+1, v, 4)
|   |   |   j = v[2] //3
|   |   |   inverte(2+1, v, 4)
|   |   |   |   j = v[3] //0
|   |   |   |   inverte(3+1, v, 4)
|   |   |   |   v[0] = 3
|   |   |   |   3,1,3,0
|   |   |   v[3] = 2
|   |   |   3,1,3,2
```

Algoritmos recursivos - Exemplo

```c
/*   0 1 2 3
    {2,1,3,0}

     0 1 2 3
    {3,1,0,2}
*/
void inverte(int i, int v[], int tam) {
  if(i<tam){
    int j = v[i];
    inverte(i+1, v, tam);
    v[j] = i;

    for(int p=0; p<tam; p++)
      printf("%d ", v[p]);
    printf("\n");
  }
}
```

```
inverte(0, v, 4)
|   j = v[0] //2
|   inverte(0+1, v, 4)
|   |   j = v[1] //1
|   |   inverte(1+1, v, 4)
|   |   |   j = v[2] //3
|   |   |   inverte(2+1, v, 4)
|   |   |   |   j = v[3] //0
|   |   |   |   inverte(3+1, v, 4)
|   |   |   |   v[0] = 3
|   |   |   |   3,1,3,0
|   |   |   v[3] = 2
|   |   |   3,1,3,2
|   |   v[1] = 1
|   |   3,1,3,2
```

Algoritmos recursivos - Exemplo

```
1  /*   0 1 2 3
2      {2,1,3,0}
3
4       0 1 2 3
5      {3,1,0,2}
6  */
7  void inverte(int i, int v[], int tam) {
8    if(i<tam){
9      int j = v[i];
10     inverte(i+1, v, tam);
11     v[j] = i;
12
13     for(int p=0; p<tam; p++)
14       printf("%d ", v[p]);
15     printf("\n");
16   }
17 }
```

```
1  inverte(0, v, 4)
2  |   j = v[0] //2
3  |   inverte(0+1, v, 4)
4  |   |   j = v[1] //1
5  |   |   inverte(1+1, v, 4)
6  |   |   |   j = v[2] //3
7  |   |   |   inverte(2+1, v, 4)
8  |   |   |   |   j = v[3] //0
9  |   |   |   |   inverte(3+1, v, 4)
10 |   |   |   |   v[0] = 3
11 |   |   |   |   3,1,3,0
12 |   |   |   v[3] = 2
13 |   |   |   3,1,3,2
14 |   |   v[1] = 1
15 |   |   3,1,3,2
16 |   v[2] = 0
17 |   3,1,0,2
```

Algoritmos recursivos - Exemplo

```c
/*    0 1 2 3
    {2,1,3,0}

      0 1 2 3
    {3,1,0,2}
*/
void inverte(int i, int v[], int tam) {
  if(i<tam){
    int j = v[i];
    inverte(i+1, v, tam);
    v[j] = i;

    for(int p=0; p<tam; p++)
      printf("%d ", v[p]);
    printf("\n");
  }
}
```

```
inverte(0, v, 4)
|   j = v[0] //2
|   inverte(0+1, v, 4)
|   |   j = v[1] //1
|   |   inverte(1+1, v, 4)
|   |   |   j = v[2] //3
|   |   |   inverte(2+1, v, 4)
|   |   |   |   j = v[3] //0
|   |   |   |   inverte(3+1, v, 4)
|   |   |   |   v[0] = 3
|   |   |   |   3,1,3,0
|   |   |   v[3] = 2
|   |   |   3,1,3,2
|   |   v[1] = 1
|   |   3,1,3,2
|   v[2] = 0
|   3,1,0,2
```

- O vetor `v[1..n]` contém uma permutação de `1..n` e a função inverte essa permutação, fazendo `v[i] = j` ser alterado para `v[j] = i`
- Inverte: guarda todos os conteúdos e, a cada retorno, realiza uma troca

Recursão

- Resolve repetidamente e volta com a solução
- Volta para o ponto de partida
- Resolve dependências entre as soluções
- Utilizado em problemas com dependência entre os valores
- Utilizado em problemas de "divisão e conquista"
  - Dividir o problema em partes
  - Resolva as partes para a solução total
- Cuidado com estouro de pilhas:
  - Garanta a condição de parada
  - Utilize técnicas como a recursão de cauda (tail call - chamada recursiva é a última instrução a ser executada) e otimizações na compilação (gcc -O2)

Recursão

```c
#include <stdio.h>

void recursiveFunction1(int num) {
  if (num > 0)
      recursiveFunction1(num - 1);
  printf("%d\n", num);
}

//tail call
void recursiveFunction2(int num) {
  printf("%d\n", num);
  if (num > 0)
      recursiveFunction2(num - 1);
}

int main() {
  //int 4 bytes -> 8 MB = 8000 KB = 8000000 B
  //recursiveFunction1(8000000);
  recursiveFunction2(8000000);
  return 0;
}
```

Recursão

```
Teste com gcc teste.c
          gcc -O1 teste.c
          gcc -O2 teste.c

Observe o assembly (quando há chamadas recursivas?)
          gcc -S teste.c
          cat teste.s
          gcc -S -O2 teste.c
          cat teste.s
```

Recursão

1. Escreva uma função recursiva que conte o número de células de uma lista encadeada.
2. Escreva uma função recursiva que imprima uma lista encadeada.
3. **Altura**. A altura de uma célula c em uma lista encadeada é a **distância entre c e o fim da lista**. Escreva uma função recursiva que calcule a altura de uma dada célula.
4. **Profundidade**. A profundidade de uma célula c em uma lista encadeada é **distância entre o início da lista e c**. Escreva uma função recursiva que calcule a profundidade de uma dada célula.
5. Escreva uma função recursiva que inverta a ordem das células de uma lista encadeada. Faça isso sem usar espaço auxiliar, apenas alterando ponteiros.