

Pesquisa em Projeto de Algoritmos



Gerencie o tempo
e tome decisões
de forma eficaz.

Programação dinâmica

Tratando-se de problemas de alta complexidade, sobretudo aqueles que envolvem a otimização de certos parâmetros, estão disponíveis diversas metodologias de programação que podem facilitar tanto o entendimento quanto o desenvolvimento da solução. Numa breve revisão, sabe-se que os problemas de otimização possuem geralmente os seguintes aspectos:

Programação dinâmica

A programação dinâmica é paradigma para projeto de algoritmos para a resolução de problemas de otimização combinatória, em particular.

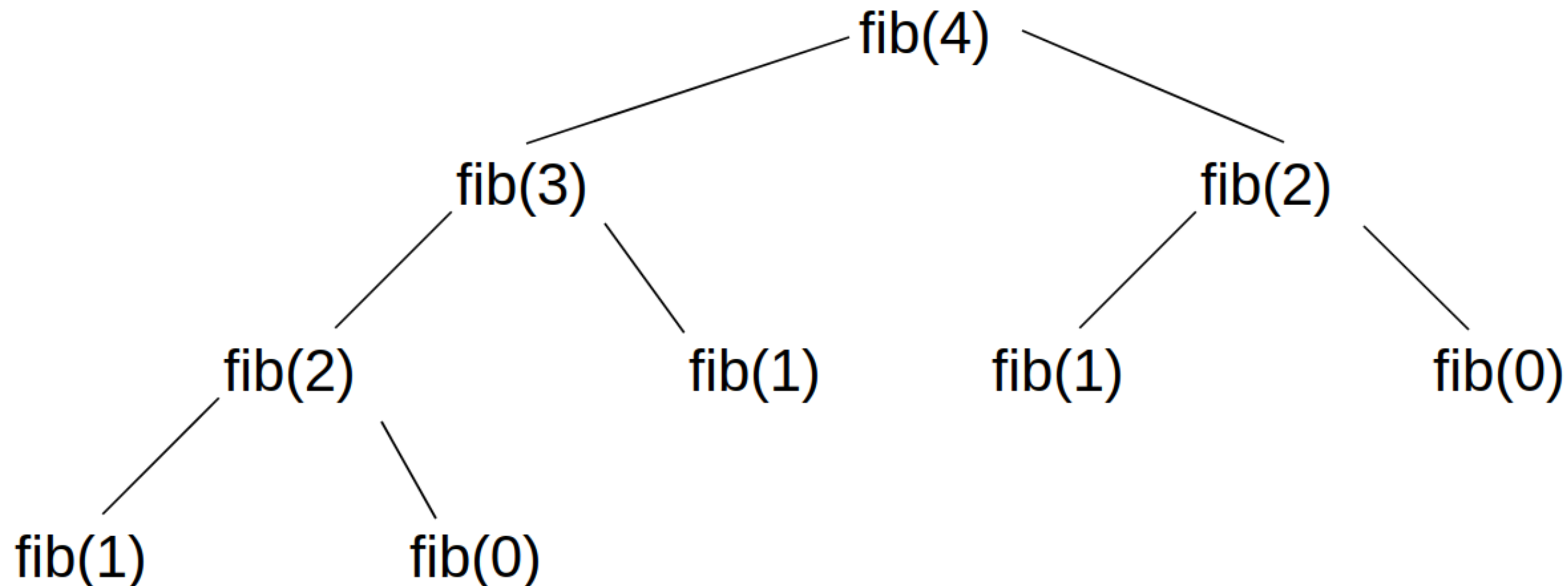
É aplicável a problemas em que a solução ótima pode ser computada a partir das soluções de seus subproblemas que, previamente computadas e memorizadas, podem ser sobrepostas para gerar a solução do problema.

Dessa forma, a programação dinâmica consiste em computar a solução de todos os possíveis subproblemas de um problema, partindo dos menores para os maiores, armazenando os resultados em uma matriz ou tabela.

Uma grande vantagem desse paradigma é que uma vez que um subproblema é resolvido e sua resposta é armazenada na matriz, a solução desse subproblema nunca mais será recalculada.

Função recursiva

Uma função recursiva $f(x)$ é uma função cuja definição envolve, de alguma forma, a si mesma. O conjunto de instâncias em que a função assume valores definidos é chamado de base da recursão. Para os cálculos da função em valores fora da base a própria função é chamada. Vamos analisar um exemplo.



Problema do troco mínimo:

Em um país "Y", tem-se como sistema de moedas os seguintes valores: 25, 21, 10, 5, 1. Para tal, ao dar um troco deve-se passar a menor quantidade de moedas.

Problema 1: Quantas moedas são necessárias para o troco de 43?

Problema 2: Quantas moedas são necessárias para o troco de 65?

Método Guloso x Recursão

```
int troco_(int valor, int *moedas, int m)
{
    int res = 0;
    for(int i = 0; i < m; i++)
    {
        while(valor >= moedas[i])
        {
            res++;
            valor -= moedas[i];
        }
    }
    return res;
}
```

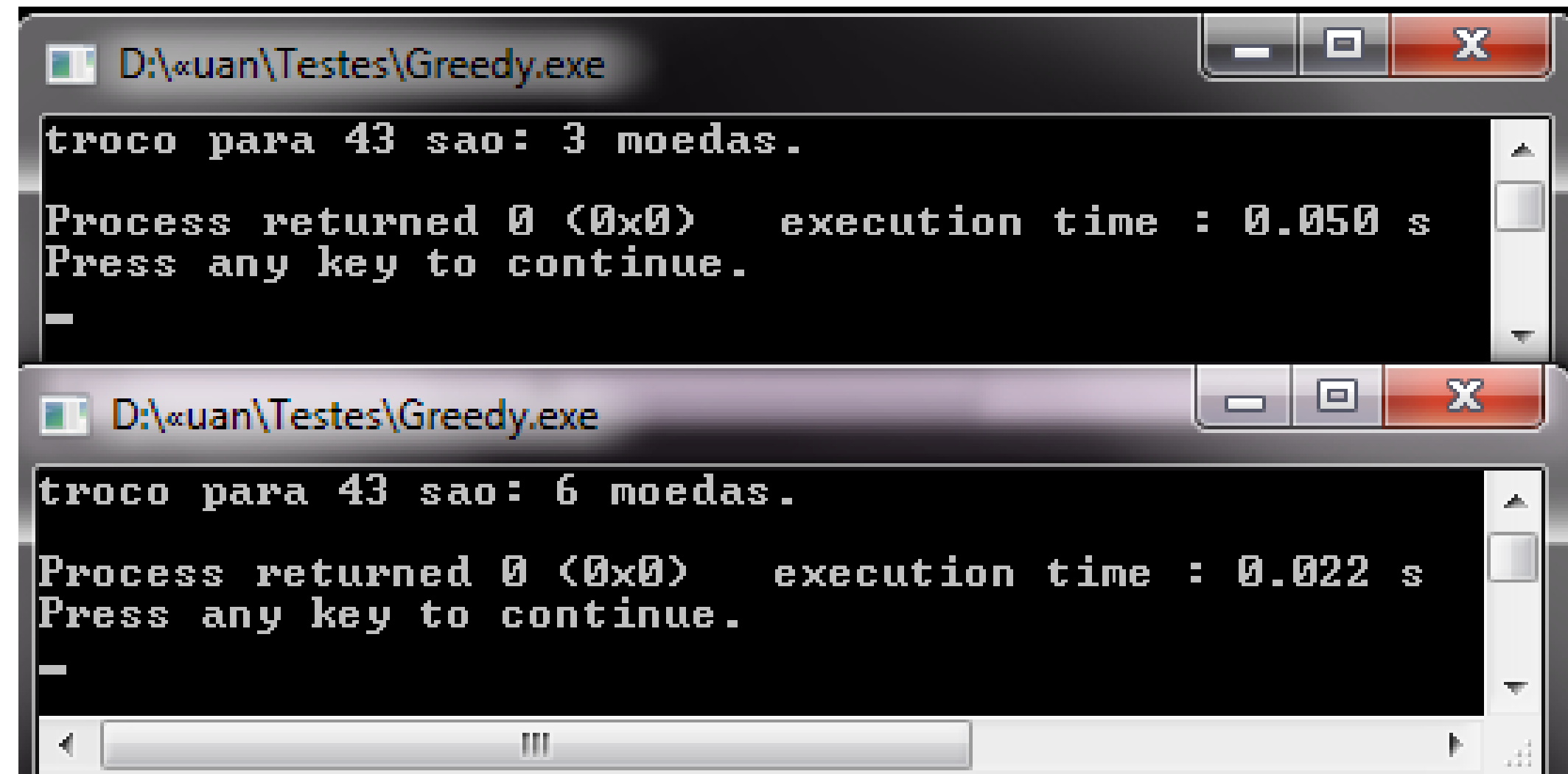
```
int troco_Recur(int valor, int *moedas, int m)
{
    int res, i=0, t=0;

    if(valor < 0)
        return -1;
    if (valor == 0){
        res = 0;
    }
    else{
        res = valor;
        for(i = 0; i < m; i++) {
            if (valor >= moedas[i])
            {
                t = troco_Recur(valor - moedas[i], moedas, m) + 1;
                if ( t < res )
                {
                    res = t;
                }
            }
        }
    }
    return res;
}
```

Recursão

X

Guloso



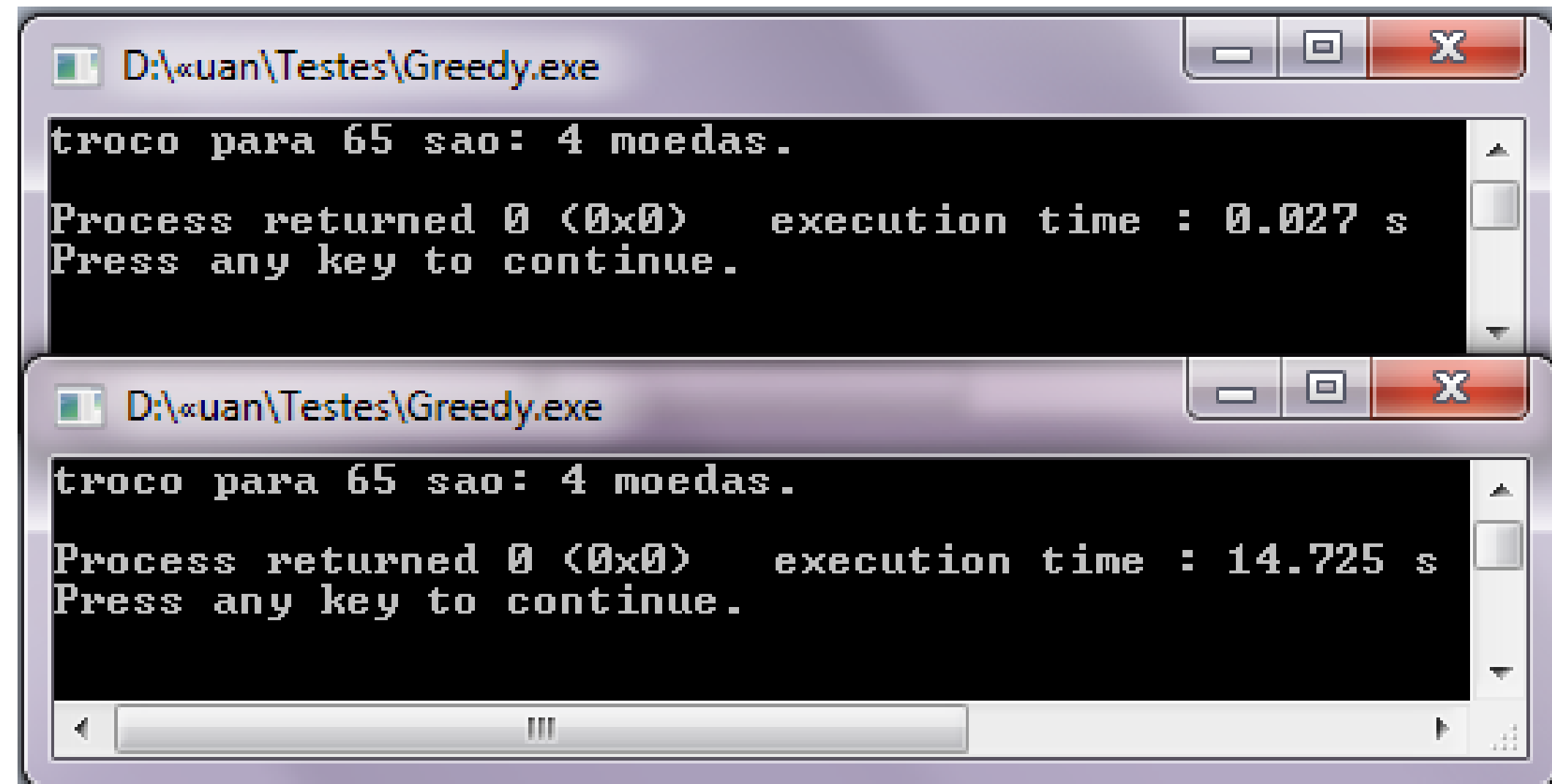
```
D:\«uan\Testes\Greedy.exe
troco para 43 sao: 3 moedas.
Process returned 0 (0x0)   execution time : 0.050 s
Press any key to continue.
_

D:\«uan\Testes\Greedy.exe
troco para 43 sao: 6 moedas.
Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.
_
```

Guloso

X

Recursão



```
D:\«uan\Testes\Greedy.exe
troco para 65 sao: 4 moedas.
Process returned 0 (0x0)   execution time : 0.027 s
Press any key to continue.

D:\«uan\Testes\Greedy.exe
troco para 65 sao: 4 moedas.
Process returned 0 (0x0)   execution time : 14.725 s
Press any key to continue.
```


Pilha:

Uma pilha, neste contexto, é o último a entrar, primeiro a sair do buffer onde você coloca os dados enquanto o programa é executado.

Quando você chama uma função em seu código, a próxima instrução após a chamada de função é armazenada na pilha e qualquer espaço de armazenamento que possa ser substituído pela chamada de função. A função que você chama pode usar mais pilha para suas próprias variáveis locais. Quando terminar, ele libera o espaço de pilha da variável local que usou e retorna à função anterior.

Stack Overflow

Geralmente, o sistema operacional e a linguagem de programação que você está usando gerenciam a pilha e ela está fora de seu controle.

Um estouro de pilha é quando você usou mais memória para a pilha do que seu programa deveria usar.

Além de boas práticas de programação, testes estáticos e dinâmicos, não há muito que você possa fazer nesses sistemas de alto nível.

Programação dinâmica

```
int troco_(int valor, int *moedas, int m)
{
    int *cache = (int*) malloc (sizeof(int) * (valor + 1));
    for(int i = 0; i <= valor; i++)
    {
        cache[i] = -1;
    }
    int res = troco_PD(cache, valor, moedas, m);
    free(cache);
    return res;
}

int main(int argc, char *argv[])
{
    int moedas[] = {25, 21, 10, 5, 1};
    int m = 5;
    int troco = 43;
    int qtd = troco_Greedy(troco, moedas, m);
    // int qtd = troco_(troco, moedas, m);
    printf("troco para %d são: %d moedas.\n", troco, qtd);
}
/*
```

```
int troco_PD(int *cache, int valor, int *moedas, int m)
{
    int res, i=0, t=0;

    if(cache[valor] == -1){
        if(valor == 0){
            res = 0;
        }
        else{
            res = valor;
            for(i = 0; i < m; i++){
                if(valor >= moedas[i])
                {
                    t = troco_PD(cache, valor - moedas[i], moedas, m) + 1;
                    if(t < res)
                    {
                        res = t;
                    }
                }
            }
        }
        cache[valor] = res;
    }
}
```

Recursão

X

Guloso

X

P. dinâmica

```
D:\«uan\Testes\Greedy.exe
troco para 43 sao: 3 moedas.
Process returned 0 (0x0)  execution time : 0.050 s
Press any key to continue.
-

D:\«uan\Testes\Greedy.exe
troco para 43 sao: 6 moedas.
Process returned 0 (0x0)  execution time : 0.022 s
Press any key to continue.
-

D:\«uan\Testes\TesteAlgo.exe
troco para 43 sao: 3 moedas.
Process returned 0 (0x0)  execution time : 0.016 s
Press any key to continue.
```

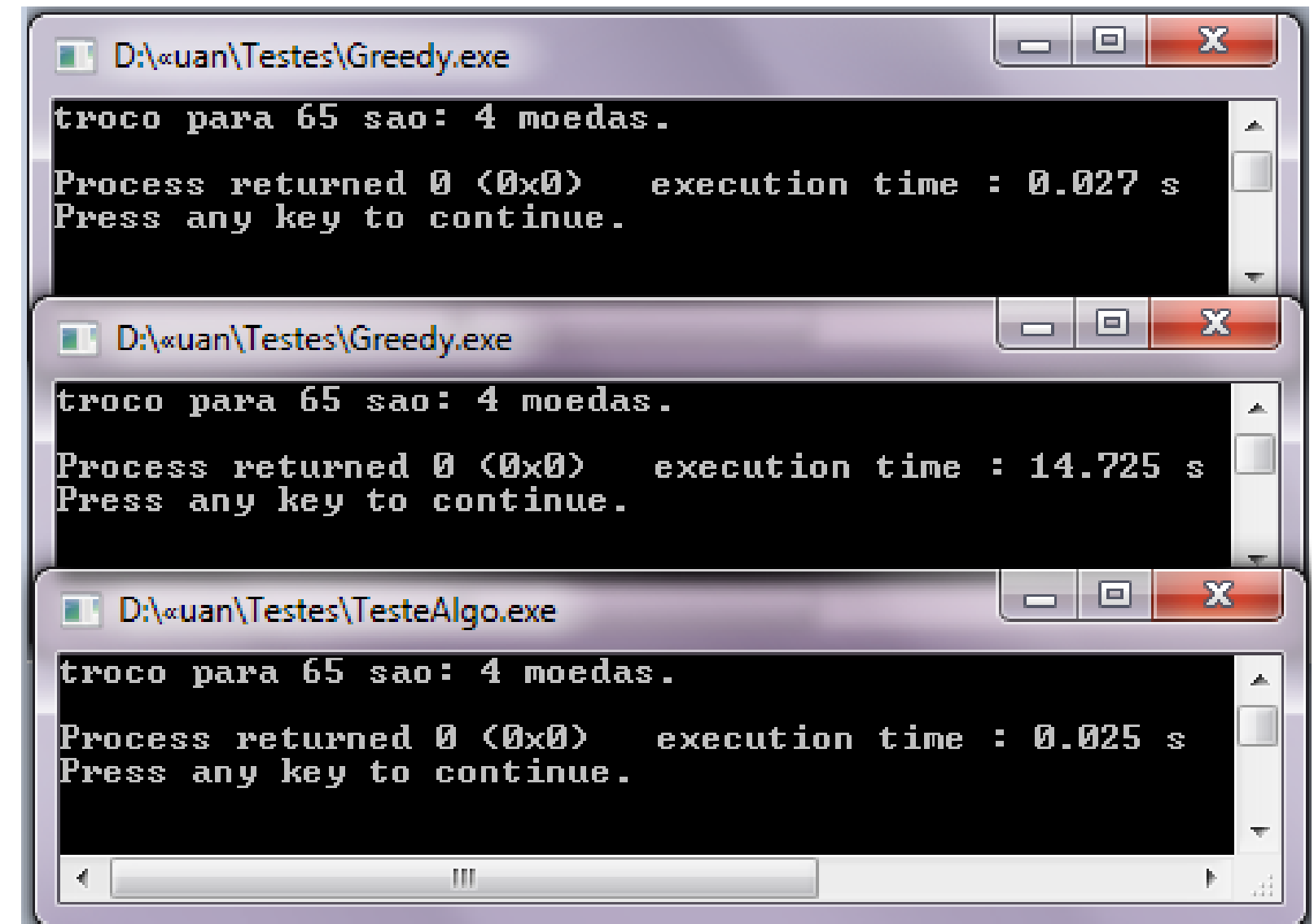
Guloso

X

Recuersão

X

P. dinâmica



Programação Dinâmica

Para que a programação dinâmica seja aplicável a um problema, duas características são necessárias:

- subestrutura ótima: um problema apresenta uma subestrutura ótima quando uma solução ótima para o problema contém soluções ótimas para seus subproblemas.**
- sobreposição de subproblemas: a sobreposição de subproblemas acontece quando um algoritmo recursivo reexamina o mesmo problema muitas vezes.**