

Pesquisa em Projeto de Algoritmos



Gerencie o tempo
e tome decisões
de forma eficaz.

Algoritmo Guloso



- **Sempre escolhe a alternativa mais "apetitosa" naquele iteração.**
- **Uma escolha que foi feita NUNCA é revista.**
- **Não se arrepende das escolhas.**
- **A escolha é feita de acordo com um critério guloso (localmente ótimo).**
- **Nem sempre dão soluções ótimas.**

Algoritmo Guloso



O problema: Para controle do valor (troco mínimo) a ser dado, deve-se passar a menor quantidade de moedas. Para tal, utiliza-se de algoritmos gulosos na tentativa de encontrar soluções ótimas.

- **Ao realizar um saque o caixa eletrônico vai disponibilizar a menor quantidade de notas de acordo com as notas disponíveis.**

Problema 1: Cliente faz uma compra de R\$ 26,00 e pagou R\$100,00. O balconista tem no caixa cédulas de R\$: 50, 23, 10, 5, 1. O troco ótimo seria?

Problema 2: Cliente faz uma compra de R\$ 7,00 e pagou R\$100,00. O balconista tem no caixa cédulas de R\$: 50, 23, 10, 5, 1. O troco ótimo seria?

Problema 3: Cliente faz uma compra de R\$ 8,00 e pagou R\$100,00. O balconista tem no caixa cédulas de R\$: 50, 23, 10, 5, 1. O troco ótimo seria?

Problema 4: Cliente faz uma compra de R\$ 31,00 e pagou R\$100,00. O balconista tem no caixa cédulas de R\$: 50, 23, 10, 5, 1. O troco ótimo seria?

Quando o método guloso funciona, geralmente o algoritmo é eficiente.

Figurativamente, a algoritmo guloso consiste em que a cada iteração será escolhido o melhor pedaço possível e não se arrependerá.

Para saber se o método guloso é eficiente, é necessário provar que o algoritmo resolve o problema.

- **Para tal, usaremos o problema do "troco minimo."**



```

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 main(int argc, char *argv[])
{
    int moedas[] = {50, 23, 10, 5, 1};
    int m = 5;
    int troco = 74;
    int qtd = troco_(troco, moedas, m);
    printf("troco para %d sao: %d moedas \n", troco, qtd);
}

/*
    {50,, 23, 1} (GULOSO)
    { 23, 23, 23, 5}
*/

```

Problema 1:

O Algoritmo guloso sempre inicia pela maior opção, no caso 50.

Problema 2:

Novamente, a opção mais apetitosa na primeira iteração é 50.

```
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 main(int argc, char *argv[])
{
    int moedas[] = {50, 23, 10, 5, 1};
    int m = 5;
    int troco = 93;
    int qtd = troco_(troco, moedas, m);
    printf("troco para %d sao: %d moedas\n", troco, qtd);
}

/*
    {50,, 23, 10, 10,} (GULOSO)
    { 23, 23, 23, 23, 1}
*/
```


Problema 4:

```
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 main(int argc, char *argv[])
{
    int moedas[] = {50, 23, 10, 5, 1};
    int m = 5;
    int troco = 93;
    int qtd = troco_(troco, moedas, m);
    printf("troco para %d sao: %d moedas\n", troco, qtd);
}

/*
    {50,, 23, 10, 10,} (GULOSO)
    { 23, 23, 23, 23, 1}
*/
*/
```



```

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 main(int argc, char *argv[])
{
    int moedas[] = {50, 23, 10, 5, 1};
    int m = 5;
    int troco = 69;
    int qtd = troco_(troco, moedas, m);
    printf("troco para %d sao: %d moedas.\n", troco, qtd);
}

/*
    {50,, 10, 5,, 1, 1, 1, 1} (GULOSO)
    { 23, 23, 23, 23}         (ÓTIMO)
*/

```

Problema 4:

Conclusão: A solução gulosa nem sempre é ótima. Sendo mais específico, a solução gulosa só é ótima quando os valores de moedas disponíveis atende os requisitos de uma sequência canônica.

A fim de se obter um resultado ótimo, utilizar programação dinâmica ou recursividade

