Técnicas e Análise de Algoritmos

Programação Dinâmica - Parte 03

Professor: **Jeremias Moreira Gomes**
E-mail: jeremias.gomes@idp.edu.br

Introdução

Problema do Troco

Problema do Troco

- Seja $C = \{c_1, c_2, \ldots, c_N\}$ uma sequência ordenada de N inteiros positivos distintos e M um inteiro positivo
  - O problema do troco consiste em determinar um vetor de inteiros não negativos $X = \{x_1, x_2, \ldots, x_N\}$ tal que

$$M = \sum_{i=1}^{N} x_i c_i \quad \text{e que a soma} \quad S = \sum_{i=1}^{N} x_i \quad \text{seja mínima}$$

Problema do Troco

- O conjunto C é chamado de conjunto de moedas
- M é o valor de um troco
- A versão informal da leitura do problema é:
  - “Qual a menor quantidade de moedas para dar o troco M?”
- Se c1 = 1, sempre há solução para qualquer valor de M
- Se C é o conjunto de moedas do sistema financeiro de algum país, esse problema pode ser resolvido por um algoritmo guloso

Algoritmo Guloso para o Problema do Troco

- Os tipos de moedas encontram-se ordenados
- Da maior para a menor, verifica a maior moeda que é menor ou igual do M (moeda ck)
  - Atribuí-se ao resultado M / c_k, subtraí-se de M o valor x_k * c_k
- Algoritmo continua até M == 0
- Tipos de moedas c_i cujo o valor não foi maior do que o M, resolvem-se como $x_i = 0$

Algoritmo Guloso para o Problema do Troco

i	c_i	x_i
5	50	
4	25	
3	10	
2	5	
1	1	

M = 74

Algoritmo Guloso para o Problema do Troco

i	c_i	x_i
5	50	1
4	25	
3	10	
2	5	
1	1	

M = 24

Algoritmo Guloso para o Problema do Troco

i	c_i	x_i
5	50	1
4	25	0
3	10	
2	5	
1	1	

M = 24

Algoritmo Guloso para o Problema do Troco

i	c_i	x_i
5	50	1
4	25	0
3	10	2
2	5	
1	1	

M = 4

Algoritmo Guloso para o Problema do Troco

i	c_i	x_i
5	50	1
4	25	0
3	10	2
2	5	0
1	1	

M = 4

Algoritmo Guloso para o Problema do Troco

i	c_i	x_i
5	50	1
4	25	0
3	10	2
2	5	0
1	1	4

M = 0

Problema do Troco - Versão gulosa

```
vector<pair<int,int>> troco(int M)
{
    vector<pair<int, int>> X;
    for (int i = 0; i < 5; i++) {
        if (moedas[i] <= M) {
            X.push_back({moedas[i], M / moedas[i]});
            M %= moedas[i];
        }
    }

    return X;
}
```

Problema com o Algoritmo Guloso

- No caso do problema do troco, o algoritmo guloso nem sempre produz a solução correta, dependendo do conjunto de moedas
  - C = {6, 5, 1}, M = 10

Incorretude do Algoritmo Guloso

- No caso do problema do troco, o algoritmo guloso nem sempre produz a solução correta, dependendo do conjunto de moedas
  - C = {6, 5, 1}, M = 10
    - Algoritmo guloso: X = {(6, 1), (1, 4)} ⇒ 5 moedas
    - Solução ótima: X = {(5, 2)} ⇒ 2 moedas
- Para uma solução correta, seguir soluções locais não leva a um ótimo global e é necessário verificar as diferentes combinações

Bases Canônicas

- Mas por que o problema acontece?
  - As bases de moedas para as quais a resolução gulosa funciona, são chamadas de “bases canônicas”
  - Exemplo: qualquer base C = {1, c_2} é canônica
- Quando as bases de moedas são canônicas, a resolução do problema pode ser feita em tempo linear (O(N)), enquanto o contrário existe um algoritmo de programação dinâmica (O(NM))
- Descobrir se uma base é canônica ou não já é outro problema de computação para ser resolvido (não tratarei nesta disciplina)

Problema do Troco (Programação Dinâmica)

- Para resolver o problema utilizando programação dinâmica
  - Seja c(m) o mínimo de moedas para um troco m
  - Se m = 0, nesse caso c(0) == 0 (caso base)
  - Caso $c_k \leq m$, então (chamado de transição de estado):
    - $c(m) = \min(c(m - c_{k1}), c(m - c_{k2}), \ldots, c(m - c_{kr})\} + 1$
    - Escolhe-se a moeda que gera a menor solução
  - M estados distintos e N transições, a complexidade O(MN)

Problema do Troco - Versão PD (top-down)

```
int troco(int m)
{
    if (m == 0) return 0;

    if (pd[m] != -1) return pd[m];

    int ans = oo;
    for (int i = 0; i < 3; i++) {
        if (moedas[i] <= m) {
            ans = min(ans, troco(m - moedas[i]) + 1);
        }
    }
    pd[m] = ans;
    return ans;
}
```

Problema do Troco (Programação Dinâmica)

- Já uma versão bottom-up para a solução do problema pode ser feita da seguinte forma:
  - A partir da versão top-down, o normal seria avaliar, para cada M, todas as moedas
  - Mas é possível inverter e montar o vetor a partir de cada moeda, avaliando todos os trocos possíveis
    - Melhora um pouco a performance essa inversão

Problema do Troco - Versão PD (bottom-up)

```
int troco(int M)
{
    for (int i = 0; i <= M; i++) pd[i] = oo;

    pd[0] = 0;

    for (int i = 0; i < 3; i++) {
        for (int m = moedas[i]; m <= M; m++) {
            pd[m] = min(pd[m], pd[m - moedas[i]] + 1);
        }

    }

    return pd[M];
}
```

Maior Subsequência Crescente (LIS)

Problema da Maior Subsequência Crescente

- Considere a sequência a = $\{a_1, a_2, ..., a_N\}$
- Uma subsequência

$$b = \{b_1, b_2, ..., b_k\} = \{a_{i1}, a_{i2}, ..., a_{ik}\}$$

de a é a **maior subsequência crescente (LIS)** se vale o seguinte:

- **$i_1 < i_2 < ... < i_k$,**
- **$b_i < b_j$ se $i < j$, e**
- **k é máximo**

LIS - Características

- O problema da maior subsequência crescente tem solução para qualquer sequência a, dado que qualquer subsequência composta por um único elemento é uma subsequência crescente
- Exemplo: a = {6, 3, 7, 1, 9, 2}
  - {6} é uma subsequência crescente
  - {6, 7} e {3, 7, 9} são subsequências crescentes
  - {7, 9, 2} é uma subsequência, mas não é crescente

LIS - Características

- O conjunto a pode ser de qualquer tipo
  - Desde que o operador < esteja definido
- Exemplo de sequência: a = {4, 1, 5, 2, 6, 3}
  - A solução (3) possui várias subsequências crescentes de mesmo tamanho: {4, 5, 6}, {1, 2, 3}, {1, 5, 6}, {1, 2, 6}, etc

Solução Quadrática para a LIS

- Uma sequência a = {a_1, a_2, ..., a_N} tem 2^N subsequências distintas
  - Busca completa só é válida para N pequeno
- O problema que pode ser resolvido com programação dinâmica:
  - Seja LIS(i) o tamanho da maior subsequência de a onde o último elemento é a_i
  - Se i = 1, então LIS(1) = 1 (caso base)

Solução Quadrática para a LIS

- O problema que pode ser resolvido com programação dinâmica:
  - Se i = 1, então LIS(1) = 1 (caso base)
  - A transição avalia todas as subsequências anteriores de a_i
    - $LIS(i) = \max\{1, LIS(a_k) + 1\}$
      - para todo $k \in [1, i)$ tal que $a_k < a_i$
- Cada transição tem custo O(N) e existem O(N) estados distintos, logo a complexidade é $O(N^2)$

LIS - Solução Quadrática com PD

```
int solver(vector<int>& V)
{
    int N = V.size();
    vector<int> pd(N, 1);

    for (int i = 1; i < N; i++) {
        for (int j = i - 1; j >= 0; j--) {
            if (V[i] > V[j]) {
                pd[i] = max(pd[i], pd[j] + 1);
            }
        }
    }

    return *max_element(pd.begin(), pd.end());
}
```

Solução Linearítimica para a LIS

- A solução quadrática ainda não é a melhor solução para o problema da LIS
- É possível programar os estados de uma forma diferente:
  - Seja LIS(k, i) o menor elemento que finaliza uma subsequência crescente de tamanho k
- Agora o caso base acontece quando i = 0, ou seja, não há elementos na subsequência

Solução Linearítimica para a LIS

- Agora, para cada i, apenas LIS(k, i) é atualizado
  - Inicialmente LIS(k, 0) = oo, para todo k $\in$ [1, N]
  - LIS(0, 0) = 0
- Cabe ressaltar que os elementos da sequência LIS(1, i), LIS(2, i), ..., encontram-se em ordem crescente
  - Assim, pode-se utilizar uma busca binária para identificar o primeiro índice j tal que LIS(j, i - 1) seja maior que a_i
  - Então, LIS(j, i) = a_i
- O tamanho da LIS será igual ao maior índice j tal que LIS(j, N) < oo

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i =$

$lis =$

1	2	3	4	5	6	7	8	9
∞	∞	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 1$

$lis =$

1	2	3	4	5	6	7	8	9
∞	∞	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 1$

$lis =$

1	2	3	4	5	6	7	8	9
4	∞	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$$
$$i = 2$$

$lis =$

1	2	3	4	5	6	7	8	9
4	∞	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$$
$$i = 2$$

$lis =$

1	2	3	4	5	6	7	8	9
4	7	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 3$

$lis =$

1	2	3	4	5	6	7	8	9
4	7	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 3$

$lis =$

1	2	3	4	5	6	7	8	9
1	7	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 4$

$lis =$

1	2	3	4	5	6	7	8	9
1	7	∞	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$$
$$i = 4$$

	1	2	3	4	5	6	7	8	9
$lis =$	1	7	9	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 5$

$lis =$

1	2	3	4	5	6	7	8	9
1	7	9	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 5$

$lis =$

1	2	3	4	5	6	7	8	9
1	3	9	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 6$

$lis =$

1	2	3	4	5	6	7	8	9
1	3	9	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$$
$$i = 6$$

	1	2	3	4	5	6	7	8	9
$lis =$	1	2	9	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 7$

$lis =$

1	2	3	4	5	6	7	8	9
1	2	9	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$$
$$i = 7$$

	1	2	3	4	5	6	7	8	9
$lis =$	1	2	6	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 8$

	1	2	3	4	5	6	7	8	9
$lis =$	1	2	6	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 8$

$lis =$

1	2	3	4	5	6	7	8	9
1	2	5	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$

$i = 9$

	1	2	3	4	5	6	7	8	9
$lis =$	1	2	5	∞	∞	∞	∞	∞	∞

Solução Linearítimica para a LIS

$$a = \{4, 7, 1, 9, 3, 2, 6, 5, 8\}$$
$$i = 9$$

	1	2	3	4	5	6	7	8	9
$lis =$	1	2	5	8	∞	∞	∞	∞	∞

LIS - Solução Linearítimica com PD

```
int solver(vector<int>& V)
{
    int N = V.size();
    vector<int> pd(N, 1);

    for (int i = 1; i < N; i++) {
        for (int j = i - 1; j >= 0; j--) {
            if (V[i] > V[j]) {
                pd[i] = max(pd[i], pd[j] + 1);
            }
        }
    }

    return *max_element(pd.begin(), pd.end());
}
```

Conclusão