Pilhas e Filas

Pilhas Monótonas

Prof. Edson Alves – UnB/FCTE

Sumário

1. Definição
2. Implementação
3. Aplicações de Pilhas Monótonas

Definição

Definição de monotonicidade

Função não-decrescente e função não-crescente

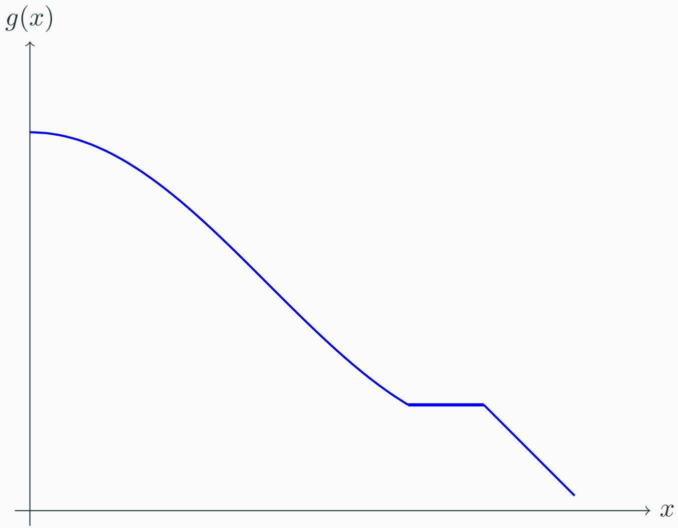
Seja $f : A \to B$ uma função. Dizemos que f é **não-decrescente** se para todo par $x, y \in A$, com $x \leq y$, temos que $f(x) \leq f(y)$.

Seja $g : A \to B$ uma função. Dizemos que g é **não-crescente** se para todo par $x, y \in A$, com $x \leq y$, temos que $g(x) \geq g(y)$.

Função monótona

Seja $h : A \to B$ uma função. Dizemos que h é **monótona** se h é não-decrescente ou não-crescente.

Exemplo de função não-decrescente



Definição de pilha monótona

Pilha monótona

Seja P uma pilha de elementos do tipo T. A pilha P é dita **monótona** se, quando extraídos todos os elementos de P, eles formam uma sequência $x_1, x_2, \ldots, x_N$, onde x_i é o elemento obtido na i-ésima extração, tais que a função $F : \mathbb{N} \to T$, com $f(i) = x_i$, é monótona.

A pilha P será **não-decrescente** se f for **não-crescente**; caso contrário, P será não-decrescente.

Inserção em pilhas monótonas

- Em pilhas monótonas é necessário manter a invariante da monotonicidade a cada inserção
- Seja P uma pilha não-decrescente e x um elemento a ser inserido em P
- Se P estiver vazia, basta inserir x em P: o invariante estará preservado
- Se P não estiver vazia, o mesmo acontece se $x \leq y$, onde y é o topo de P
- Contudo, se $x > y$, é preciso remover y antes da inserção de x
- Após a remoção de y, é preciso confrontar x com o novo topo até que x possa ser inserido em P

Exemplo de inserções em uma pilha não-decrescente

$a_n =$	1	4	3	4	2	1	3

Exemplo de inserções em uma pilha não-decrescente

$a_n =$

1	4	3	4	2	1	3

Exemplo de inserções em uma pilha não-decrescente

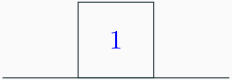
$a_n =$

1	4	3	4	2	1	3

Exemplo de inserções em uma pilha não-decrescente

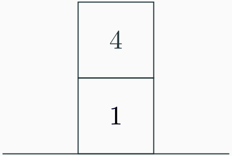
$a_n =$

1	4	3	4	2	1	3



Exemplo de inserções em uma pilha não-decrescente

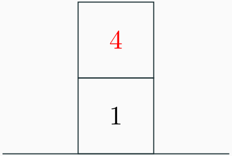
$$a_n = \begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 4 & 3 & 4 & 2 & 1 & 3 \\ \hline \end{array}$$



Exemplo de inserções em uma pilha não-decrescente

$a_n =$

1	4	3	4	2	1	3



Exemplo de inserções em uma pilha não-decrescente

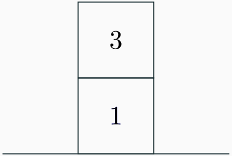
$a_n =$

1	4	3	4	2	1	3

1

Exemplo de inserções em uma pilha não-decrescente

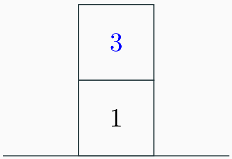
$$a_n = \begin{array}{|c|c|c|c|c|c|c|}\hline 1 & 4 & 3 & 4 & 2 & 1 & 3 \\ \hline \end{array}$$



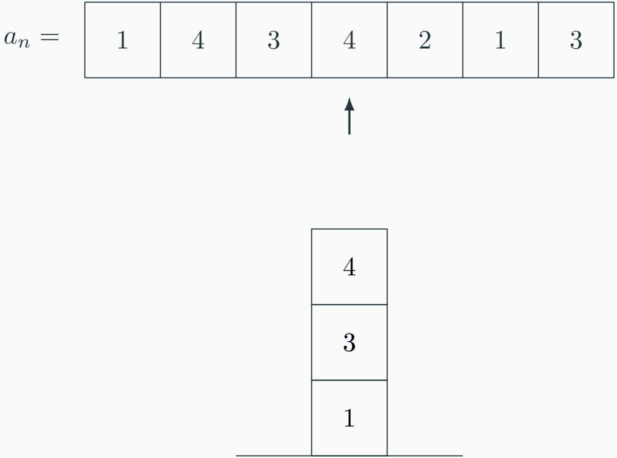
Exemplo de inserções em uma pilha não-decrescente

$a_n =$

1	4	3	4	2	1	3



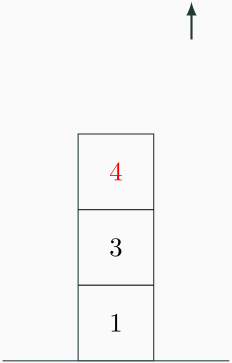
Exemplo de inserções em uma pilha não-decrescente



Exemplo de inserções em uma pilha não-decrescente

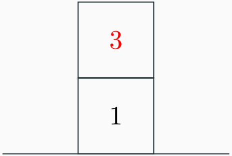
$a_n =$

1	4	3	4	2	1	3



Exemplo de inserções em uma pilha não-decrescente

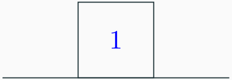
$$a_n = \boxed{1} \boxed{4} \boxed{3} \boxed{4} \boxed{2} \boxed{1} \boxed{3}$$



Exemplo de inserções em uma pilha não-decrescente

$a_n =$

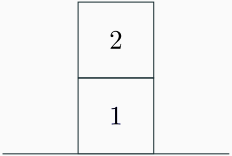
1	4	3	4	2	1	3



Exemplo de inserções em uma pilha não-decrescente

$a_n =$

1	4	3	4	2	1	3



Exemplo de inserções em uma pilha não-decrescente

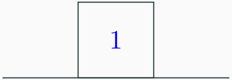
$$a_n = \begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 4 & 3 & 4 & 2 & 1 & 3 \\ \hline \end{array}$$

2
1

Exemplo de inserções em uma pilha não-decrescente

$a_n =$

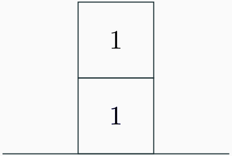
1	4	3	4	2	1	3



Exemplo de inserções em uma pilha não-decrescente

$a_n =$

1	4	3	4	2	1	3



Implementação

Implementação de uma pilha não-decrescente em C++

```cpp
template<typename T>
class MonoStack {
public:
    void push(const T& x)
    {
        while (not st.empty() and st.top() > x)
            st.pop();

        st.emplace(x);
    }

    void pop() { st.pop(); };
    auto top() const { return st.top(); }
    bool empty() const { return st.empty(); }

private:
    stack<T> st;
};
```

Implementação de uma pilha não-decrescente em C++

```cpp
template<typename T>
ostream& operator<<(ostream& os, const MonoStack<T>& ms)
{
    auto temp(ms);

    while (not temp.empty())
    {
        cout << temp.top() << ' ';
        temp.pop();
    }

    return os;
}
```

Implementação de uma pilha não-decrescente em C++

```cpp
int main()
{
    vector<int> as { 1, 4, 3, 4, 2, 1, 3 };
    MonoStack<int> ms;

    for (auto& a : as)
    {
        ms.push(a);
        cout << ms << '\n';
    }

    return 0;
}
```

Aplicações de Pilhas Monótonas

Maior elemento à esquerda (ou a direita)

Definição

Seja $a_1, a_2, \ldots, a_N$ uma sequência de elementos. O **maior elemento à esquerda (à direita) de** a_i, se existir, é o elemento a_j tal que j é o maior (menor) índice tal que $j < i$ ($j > i$) e $a_j > a_i$.

Maior elemento à esquerda em $O(N)$

- É possível determinar o maior elemento à esquerda para todos os elementos de uma sequência $a_1, a_2, \ldots, a_N$ em $O(N^2)$ por meio de uma busca completa
- Para cada índice i, é preciso avaliar todos os elementos a_j, com $j = 1, 2, \ldots, i-1$
- Contudo, é possível determinar estes valores em $O(N)$ com uma modificação no método de inserção de uma pilha não-crescente
- A inserção em uma pilha não-crescente ocorre em duas etapas: manutenção do invariante e inserção do novo elemento
- Finalizada a manutenção do invariante, os elementos que restam na pilha são todos maiores do que a_i e o elemento do topo será o maior elemento à esquerda de a_i
- Em algumas implementações são mantidos os índices e não os valores da sequência propriamente ditos (ou pares com ambas informações)

Implementação do maior elemento à esquerda em C++

```cpp
template<typename T>
class MonoStack {
public:
    MonoStack(int start_idx = 0, inc = 1) : pos(start_idx), invalid(start_idx - inc) {}

    int push(const T& x) {
        while (not st.empty() and st.top().second <= x)
            st.pop();

        auto i = st.empty() ? invalid : st.top().first;
        st.emplace(pos, x);
        pos += inc;

        return i;
    }

    void pop() { st.top(); };
    auto top() const { return st.top(); }
    bool empty() const { return st.empty(); }
```

Implementação do maior elemento à esquerda em C++

```cpp
private:
    stack<pair<int, T>> st;
    int pos, invalid;
};

auto pge(const vector<int>& xs)
{
    MonoStack<int> ms;
    vector<int> ans;

    for (auto x : xs)
        ans.emplace_back(ms.push(x));

    return ans;
}
```

Implementação do maior elemento à esquerda em C++

```cpp
int main()
{
    vector<int> as { 1, 4, 3, 4, 2, 1, 3 };
    auto is = pge(as);

    cout << "i:    ";
    for (size_t i = 0; i < as.size(); ++i)
        cout << setw(2) << setfill(' ') << i << (i + 1 == as.size() ? '\n' : ' ');

    cout << "as:   ";
    for (size_t i = 0; i < as.size(); ++i)
        cout << setw(2) << setfill(' ') << as[i] << (i + 1 == as.size() ? '\n' : ' ');

```

Maior elemento à direita em $O(N)$

- É possível determinar o maior elemento à direita usando estratégia semelhante
- Basta inserir os elementos da direita para a esquerda em uma pilha não-crescente
- Construa a instância da classe `MonoStack` com a seguinte expressão:

```
MonoStack<int> ms(N - 1, -1);
```

- Também é possível computar ambos elementos de uma só vez, caso os elementos da sequência sejam todos distintos
- Neste caso, o elemento mais à direita de a_i será o elemento a_j que remove a_i da pilha na manutenção do invariante
- Em uma implementação direta, sem uso de classes, é possível obter ambos vetores (pge – *previous greater element* – e nge – *next greater element*) com uma única chamada de função

Implementação do maior elemento à esquerda e à direita em C++

```cpp
auto ge(const vector<int>& as)
{
    auto N = (int) as.size();
    vector<int> pge(N, -1), nge(N, N);
    stack<int> st;

    for (int i = 0; i < N; ++i) {
        while (not st.empty() and as[st.top()] <= as[i]) {
            nge[st.top()] = i;
            st.pop();
        }

        if (not st.empty())
            pge[i] = st.top();

        st.emplace(i);
    }

    return make_pair(pge, nge);
}
```

Maior elemento à esquerda e Programação Dinâmica

- As ideias apresentadas até o momento permitem o desenvolvimento de um algoritmo de programação dinâmica para determinar o maior elemento à esquerda de todos os elementos da sequência $a_1, a_2, \ldots, a_N$ em $O(N)$
- Seja $pge(i)$ o índice do maior elemento à esquerda de a_i, ou 0, caso não exista
- O caso base acontece quando $i = 1$: como não há elementos à esquerda de a_1, vale que $pge(1) = 0$
- Em relação à transição, há duas possibilidades:
    1. se $a_{i-1} > a_i$, então $pge(a_i) = i - 1$
    2. caso contrário, $pge(a_i)$ será o menor elemento da sequência $pge(a_{i-1}), pge^2(a_{i-1}), \ldots, pge^k(a_{i-1}), \ldots, 0$

Implementação do maior elemento usando DP

```
auto pge(const vector<int>& as)
{
    auto N = (int) as.size();
    vector<int> dp(N, -1);

    for (int i = 1; i < N; ++i)
    {
        auto j = i - 1;

        while (j >= 0 and as[j] <= as[i])
            j = dp[j];

        dp[i] = j;
    }

    return dp;
}
```

Implementação do maior elemento usando DP

```
auto nge(const vector<int>& as)
{
    auto N = (int) as.size();
    vector<int> dp(N, N);

    for (int i = N - 2; i >= 0; --i)
    {
        auto j = i + 1;

        while (j < N and as[j] <= as[i])
            j = dp[j];

        dp[i] = j;
    }

    return dp;
}
```

Soma dos elementos máximos de todos os intervalos

Definição

Seja $a_1, a_2, \ldots, a_N$ uma sequência. O problema da **soma dos elementos máximos de todos os intervalos** da sequência consiste em determinar a soma

$$S(a_N) = \sum_{i=1}^{N} \sum_{j=i}^{N} \max(a_i, a_{i+1}, \ldots, a_j)$$

Soma dos elementos máximos de todos os intervalos em $O(N)$

- Como há $N(N+1)/2$ intervalos válidos com índices em $[1, N]$, a princípio parece que o problema só admite soluções com complexidade $O(N^2)$ ou maior
- Contudo, conhecidos os vetores pge e nge (computados por pilhas monótonas ou por DP), é possível resolver este problema em $O(N)$
- Para tanto, é preciso fazer um ajuste em um dos dois vetores para que, ao invés do maior elemento, seja o elemento maior ou igual
- Assuma, sem perda de generalidade, que o ajuste seja feito à direita, de modo que obtenhamos o vetor ngee (*next greater or equal element*)
- Seja $L_i = pge(i)$ e $R_i = ngee(i)$ para um elemento a_i qualquer

Soma dos elementos máximos de todos os intervalos em $O(N)$

- Temos que a_i será o elemento máximo de todos os intervalos $[j, k]$ tais que $L_i < j \leq i$ e $i \leq k < R_i$
- O ajuste no vetor à direita é feito para evitar a duplicidade de intervalos na contagem: a_i será o máximo de todos os intervalos cujo máximo é a_i e a_i é o maior elemento, à direita do intervalo, que tem este valor
- Assim, a_i contribuirá, para a soma, com a quantia $a_i(i - L_i)(R_i - i)$
- Deste modo,

$$S(a_N) = \sum_{i=1}^{N}\sum_{j=i}^{N} \max(a_i, a_{i+1}, \ldots, a_j) = \sum_{i=1}^{N} a_i(i - L_i)(R_i - i)$$

Implementação da soma dos máximos dos intervalos em C++

```cpp
using ll = long long;

auto pge(const vector<ll>& as)
{
    auto N = (int) as.size();
    vector<ll> dp(N, -1);

    for (int i = 1; i < N; ++i)
    {
        ll j = i - 1;

        while (j >= 0 and as[j] <= as[i])
            j = dp[j];

        dp[i] = j;
    }

    return dp;
}
```

Implementação da soma dos máximos dos intervalos em C++

```cpp
auto ngee(const vector<ll>& as)
{
    auto N = (int) as.size();
    vector<ll> dp(N, N);

    for (int i = N - 2; i >= 0; --i)
    {
        ll j = i + 1;

        while (j < N and as[j] < as[i])
            j = dp[j];

        dp[i] = j;
    }

    return dp;
}
```

Implementação da soma dos máximos dos intervalos em C++

```cpp
auto sum_of_subarray_maximums(const vector<ll>& as)
{
    auto N = (int) as.size();
    auto L = pge(as), R = ngee(as);
    ll sum = 0;

    for (int i = 0; i < N; ++i)
        sum += as[i]*(i - L[i])*(R[i] - i);

    return sum;
}
```

Referências

1. **AlgoMonster**. Monotonic Stack Data Structure Explained, acesso em 14/08/2025.
2. **animeshf**. Sum of all subarray maximums, acesso em 15/08/2025.
3. **GeeksForGeeks**. Introduction to Monotonic Stack - Data Structure and Algorithm Tutorials, acesso em 14/08/2025.
4. **YouKn0wwho Academy**. 124. Monotonic Stack: All Nearest Smaller Values and All Subarray Maximum/Minimum, aceso em 12/08/2025.