

Aula 02 – Custo de um Algoritmo e Complexidade

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023

Projeto de Algoritmos

- **Projetamos algoritmos para resolver problemas**

Projeto de Algoritmos

- **Projetamos algoritmos para resolver problemas**
- **Problema:** preciso verificar se uma dada pessoa está na lista de aprovados do vestibular

Projeto de Algoritmos

- **Projetamos algoritmos para resolver problemas**
- **Problema:** preciso verificar se uma dada pessoa está na lista de aprovados do vestibular
- **Solução:** algoritmo que realiza uma busca sequencial

Projeto de Algoritmos

- **Projetamos algoritmos para resolver problemas**
- **Problema:** preciso verificar se uma dada pessoa está na lista de aprovados do vestibular
- **Solução:** algoritmo que realiza uma busca sequencial
- **Problema:** preciso verificar se uma pessoa está na lista ordenada de cidadãos brasileiros

Projeto de Algoritmos

- **Projetamos algoritmos para resolver problemas**
- **Problema:** preciso verificar se uma dada pessoa está na lista de aprovados do vestibular
- **Solução:** algoritmo que realiza uma busca sequencial
- **Problema:** preciso verificar se uma pessoa está na lista ordenada de cidadãos brasileiros
- **Solução:** algoritmo que realiza a busca binária

Projeto de Algoritmos

- Analisar um algoritmo significa prever os recursos de que ele vai precisar

Projeto de Algoritmos

- Analisar um algoritmo significa prever os recursos de que ele vai precisar
 - No tempo: quanto tempo vai demorar para encontrar a solução do problema

Projeto de Algoritmos

- Analisar um algoritmo significa prever os recursos de que ele vai precisar
 - No tempo: quanto tempo vai demorar para encontrar a solução do problema
 - No espaço: quanto de memória será necessário para encontrar a solução

Projeto de Algoritmos

- Analisar um algoritmo significa prever os recursos de que ele vai precisar
 - No tempo: quanto tempo vai demorar para encontrar a solução do problema
 - No espaço: quanto de memória será necessário para encontrar a solução
- Projetar algoritmos implica analisar o seu comportamento conforme essas variáveis

Projeto de Algoritmos

- Analisar um algoritmo significa prever os recursos de que ele vai precisar
 - No tempo: quanto tempo vai demorar para encontrar a solução do problema
 - No espaço: quanto de memória será necessário para encontrar a solução
- Projetar algoritmos implica analisar o seu comportamento conforme essas variáveis
- O estudo da eficiência de algoritmos (seja no tempo ou no espaço) é chamado análise de algoritmos

Temos dois problemas distintos:

- Análise de um algoritmo particular

Temos dois problemas distintos:

- Análise de um algoritmo particular
 - Qual é o custo de usar um dado algoritmo para resolver um problema específico?

Temos dois problemas distintos:

- Análise de um algoritmo particular
 - Qual é o custo de usar um dado algoritmo para resolver um problema específico?
 - Respondido com base numa análise do número de vezes que cada parte desse algoritmo vai ser executada
 - Ou seja, do número de operações executadas

Temos dois problemas distintos:

- Análise de um algoritmo particular
 - Qual é o custo de usar um dado algoritmo para resolver um problema específico?
 - Respondido com base numa análise do número de vezes que cada parte desse algoritmo vai ser executada
 - Ou seja, do número de operações executadas
 - Seguida do estudo de quanto de memória será necessária

Temos dois problemas distintos:

- Análise de uma classe de algoritmos

Temos dois problemas distintos:

- Análise de uma classe de algoritmos
 - Qual é o algoritmo de menor custo possível para resolver um problema específico?

Temos dois problemas distintos:

- Análise de uma classe de algoritmos
 - Qual é o algoritmo de menor custo possível para resolver um problema específico?
 - Isto implica investigar toda uma família de algoritmos

Temos dois problemas distintos:

- Análise de uma classe de algoritmos
 - Qual é o algoritmo de menor custo possível para resolver um problema específico?
 - Isto implica investigar toda uma família de algoritmos
 - Buscamos identificar um que seja o melhor possível

Temos dois problemas distintos:

- Análise de uma classe de algoritmos
 - Qual é o algoritmo de menor custo possível para resolver um problema específico?
 - Isto implica investigar toda uma família de algoritmos
 - Buscamos identificar um que seja o melhor possível
 - Ao encontrá-lo, seu custo será uma medida da dificuldade inerente para resolver problemas da mesma classe.

Temos dois problemas distintos:

- Isso também significa colocar limites para a complexidade dos algoritmos:

Temos dois problemas distintos:

- Isso também significa colocar limites para a complexidade dos algoritmos:
- Exemplo: podemos estimar o número mínimo de comparações necessárias para ordenar n números por meio de comparações sucessivas

Temos dois problemas distintos:

- Isso também significa colocar limites para a complexidade dos algoritmos:
 - Exemplo: podemos estimar o número mínimo de comparações necessárias para ordenar n números por meio de comparações sucessivas
 - Logo, nenhum algoritmo vai fazer melhor que isto \Rightarrow menor custo possível

Temos dois problemas distintos:

- Isso também significa colocar limites para a complexidade dos algoritmos:
 - Exemplo: podemos estimar o número mínimo de comparações necessárias para ordenar n números por meio de comparações sucessivas
 - Logo, nenhum algoritmo vai fazer melhor que isto \Rightarrow menor custo possível
 - Menor custo possível \Rightarrow medida de dificuldade.

Temos dois problemas distintos:

- Isso também significa colocar limites para a complexidade dos algoritmos:
 - Exemplo: podemos estimar o número mínimo de comparações necessárias para ordenar n números por meio de comparações sucessivas
 - Logo, nenhum algoritmo vai fazer melhor que isto \Rightarrow menor custo possível
 - Menor custo possível \Rightarrow medida de dificuldade.
- Se o custo do algoritmo A for o menor custo possível, então A é **ótimo**.

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

1. Medição direta do tempo de execução em um computador real

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

1. Medição direta do tempo de execução em um computador real

- Problemas: depende do compilador; depende do hardware;

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

1. Medição direta do tempo de execução em um computador real

- Problemas: depende do compilador; depende do hardware;
- Medidas de tempo podem ser influenciadas pela memória disponível.

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

2. Medir com base em um computador ideal, em que cada instrução tem seu custo bem definido

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

2. Medir com base em um computador ideal, em que cada instrução tem seu custo bem definido
 - Calcula-se o número de vezes que cada operação é realizada

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

2. Medir com base em um computador ideal, em que cada instrução tem seu custo bem definido
 - Calcula-se o número de vezes que cada operação é realizada
 - Com uma tabela com o custo de cada tipo de operação, é possível calcular o custo total.

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

2. Medir com base em um computador ideal, em que cada instrução tem seu custo bem definido
 - Calcula-se o número de vezes que cada operação é realizada
 - Com uma tabela com o custo de cada tipo de operação, é possível calcular o custo total.
 - Problemas: é algo bastante trabalhoso e o custo de cada operação varia de acordo com o hardware, compilador etc;

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

3. Calcula-se o número de vezes que as **operações mais significativas** são executadas, ignorando as demais operações

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

3. Calcula-se o número de vezes que as **operações mais significativas** são executadas, ignorando as demais operações

- É a solução mais comumente usada e suficientemente precisa, especialmente quando queremos analisar a complexidade assintótica;

Complexidade

Como medir o custo de um algoritmo? Como comparar o custo de vários algoritmos que resolvem um problema?

3. Calcula-se o número de vezes que as **operações mais significativas** são executadas, ignorando as demais operações

- É a solução mais comumente usada e suficientemente precisa, especialmente quando queremos analisar a complexidade assintótica;
- Exemplo: na Busca ou na Ordenação \Rightarrow número de comparações

Função de Complexidade

- Para medir o custo de execução de um algoritmo, definimos uma função de custo ou complexidade $f(n)$:

Função de Complexidade

- Para medir o custo de execução de um algoritmo, definimos uma função de custo ou complexidade $f(n)$:
- $f(n)$ é a medida do tempo ou espaço necessário para executar um algoritmo para uma entrada de tamanho n

Função de Complexidade

- Para medir o custo de execução de um algoritmo, definimos uma função de custo ou complexidade $f(n)$:
 - $f(n)$ é a medida do tempo ou espaço necessário para executar um algoritmo para uma entrada de tamanho n
 - Se for de tempo, então $f(n)$ é chamada de função de complexidade de tempo ou temporal do algoritmo.

Função de Complexidade

- Para medir o custo de execução de um algoritmo, definimos uma função de custo ou complexidade $f(n)$:
 - $f(n)$ é a medida do tempo ou espaço necessário para executar um algoritmo para uma entrada de tamanho n
 - Se for de tempo, então $f(n)$ é chamada de função de complexidade de tempo ou temporal do algoritmo.
 - Se for da memória necessária (espaço) para executar o algoritmo, então $f(n)$ é a função de complexidade espacial.

Função de Complexidade

- Se nada for dito, entende-se $f(n)$ como complexidade de tempo.

Função de Complexidade

- Se nada for dito, entende-se $f(n)$ como complexidade de tempo.
- Lembre que isso não representa o tempo diretamente, mas o número de vezes que determinada operação, considerada relevante, é executada.

Função de Complexidade

- Se nada for dito, entende-se $f(n)$ como complexidade de tempo.
 - Lembre que isso não representa o tempo diretamente, mas o número de vezes que determinada operação, considerada relevante, é executada.
- Mas e isso funciona?
- Por que não condicionar apenas ao hardware, ou o tempo utilizado?

Função de Complexidade

- Se nada for dito, entende-se $f(n)$ como complexidade de tempo.
 - Lembre que isso não representa o tempo diretamente, mas o número de vezes que determinada operação, considerada relevante, é executada.
- Mas e isso funciona?
- Por que não condicionar apenas ao hardware, ou o tempo utilizado?
 - Porque não conseguiremos garantir as mesmas condições para toda e qualquer entrada

Exemplo

- Suponha dois algoritmos com duas funções de complexidade:
 - $f_1(n) = c_1 n^2$
 - $f_2(n) = c_2 \log_{10}(n)$

Exemplo

- Suponha dois algoritmos com duas funções de complexidade:
 - $f_1(n) = c_1 n^2$
 - $f_2(n) = c_2 \log_{10}(n)$
- Suponha dois computadores:
 - A: 1.000.000.000 de instruções por segundo
 - B: 10.000.000 de instruções por segundo (100 vezes mais lento)

Função de Complexidade

Exemplo

- Agora suponha dois compiladores^a
 - Um ótimo, usado no primeiro algoritmo, resultando em uma constante $c_1 = 2$ ($f_1(n) = 2n^2$)
 - Um bem pior, usado no segundo algoritmo, resultando em uma $c_2 = 50$ ($f_2(n) = 50\log_{10}(n)$)

^aComo veremos, há várias coisas que podem influenciar nas constantes.

Função de Complexidade

Exemplo

- Agora suponha dois compiladores^a
 - Um ótimo, usado no primeiro algoritmo, resultando em uma constante $c_1 = 2$ ($f_1(n) = 2n^2$)
 - Um bem pior, usado no segundo algoritmo, resultando em uma $c_2 = 50$ ($f_2(n) = 50\log_{10}(n)$)
- Fazemos então o algoritmo f_1 rodar no computador mais rápido e o f_2 no mais lento
 - f_1 não somente tem a máquina mais rápida, como o melhor compilador
 - f_2 ficou com o pior dos dois mundos

^aComo veremos, há várias coisas que podem influenciar nas constantes.

Função de Complexidade

Exemplo

- Para uma entrada de 1.000 elementos:
 - $F_1 = 2 \times (1.000)^2 = 2.000.000$ instruções
 $2.000.000 \div 1.000.000.000 = 2$ ms

Exemplo

- Para uma entrada de 1.000 elementos:
 - $F_1 = 2 \times (1.000)^2 = 2.000.000$ instruções
 $2.000.000 \div 1.000.000.000 = 2$ ms
 - $f_2 = 50 \times \log_{10}(1.000) = 150$ instruções
 $150 \div 10.000.000 = 0,015$ ms

Função de Complexidade

Exemplo

- Para uma entrada de 1.000 elementos:
 - $F_1 = 2 \times (1.000)^2 = 2.000.000$ instruções
 $2.000.000 \div 1.000.000.000 = 2$ ms
 - $f_2 = 50 \times \log_{10}(1.000) = 150$ instruções
 $150 \div 10.000.000 = 0,015$ ms
- Mesmo tendo o pior compilador e a pior máquina, para apenas 1.000 elementos já rodou ≈ 133 vezes mais rápido

Função de Complexidade – Cálculo

Exemplo

- Encontrar o maior elemento de um arranjo de inteiros A

```
int maxArranjo(int A[],int n){  
    int i, max;  
    max = A[0];  
    for(i=1; i<n; i++)  
        if(max < A[i])  
            max = A[i];  
    return max;  
}
```

Função de Complexidade – Cálculo

Exemplo

```
int maxArranjo(int A[],
               int n){
    int i, max;
    max = A[0];
    for(i=1; i<n; i++)
        if(max < A[i])
            max = A[i];
    return max;
}
```

- Seja $f(n)$ uma função de complexidade
 - $f(n)$ é o número de comparações para um arranjo A de tamanho n
- Como seria $f(n)$?

Função de Complexidade – Cálculo

Exemplo

```
int maxArranjo(int A[],  
               int n){  
    int i, max;  
    max = A[0];  
    for(i=1; i<n; i++)  
        if(max < A[i])  
            max = A[i];  
    return max;  
}
```

- Seja $f(n)$ uma função de complexidade
 - $f(n)$ é o número de comparações para um arranjo A de tamanho n
- Como seria $f(n)$?
 - $f(n) = n - 1$, para $n > 0$.

Função de Complexidade – Cálculo

Exemplo

```
int maxArranjo(int A[],  
               int n){  
    int i, max;  
    max = A[0];  
    for(i=1; i<n; i++)  
        if(max < A[i])  
            max = A[i];  
    return max;  
}
```

- Seja $f(n)$ uma função de complexidade
 - $f(n)$ é o número de comparações para um arranjo A de tamanho n
- Como seria $f(n)$?
 - $f(n) = n - 1$, para $n > 0$.
- Será que o algoritmo apresentado é ótimo?

Exemplo – Prova

- Teorema:
 - Qualquer algoritmo para encontrar o maior elemento de um conjunto de n elementos, $n \geq 1$, faz ao menos $n - 1$ comparações
- Prova:
 - Cada um dos $n - 1$ elementos tem que ser verificado, por meio de comparações, que é menor do que algum outro elemento. Logo, $n - 1$ comparações são necessárias.
 - Como `maxArranjo` possui complexidade igual ao limite inferior de custo, então seu algoritmo é ótimo.

Função de Complexidade

- Normalmente, a medida de custo de execução depende do tamanho da entrada

Função de Complexidade

- Normalmente, a medida de custo de execução depende do tamanho da entrada
 - Mas este não é o único fato que influencia o custo

Função de Complexidade

- Normalmente, a medida de custo de execução depende do tamanho da entrada
 - Mas este não é o único fato que influencia o custo
- O tipo de entrada pode também influenciar o custo

Função de Complexidade

- Normalmente, a medida de custo de execução depende do tamanho da entrada
 - Mas este não é o único fato que influencia o custo
- O tipo de entrada pode também influenciar o custo
 - No caso de `maxArranjo()`, a entrada não influencia
 - Para um algoritmo de ordenação, é possível que menos tempo será gasto se a entrada já estiver quase ordenada

Função de Complexidade

- Normalmente, a medida de custo de execução depende do tamanho da entrada
 - Mas este não é o único fato que influencia o custo
- O tipo de entrada pode também influenciar o custo
 - No caso de `maxArranjo()`, a entrada não influencia
 - Para um algoritmo de ordenação, é possível que menos tempo será gasto se a entrada já estiver quase ordenada
- Considere um outro método para obter o máximo e o mínimo de um arranjo.

Função de Complexidade

Exemplo

- Qual a função de complexidade de maxMin?

```
void minMax(int A[], int n) {
    int i, min, max;
    min = A[0];
    max = A[0];
    for(i=1; i < n; i++) {
        if(max < A[i]) max = A[i];
        else
            if(min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

Função de Complexidade

Exemplo

- Qual a função de complexidade de maxMin?
- Depende:
 - Se o arranjo já estiver ordenado em ordem crescente:

```
void minMax(int A[], int n) {
    int i, min, max;
    min = A[0];
    max = A[0];
    for(i=1; i < n; i++) {
        if(max < A[i]) max = A[i];
        else
            if(min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

Função de Complexidade

Exemplo

- Qual a função de complexidade de maxMin?
- Depende:
 - Se o arranjo já estiver ordenado em ordem crescente:

$$f(n) = n - 1$$

```
void minMax(int A[], int n) {
    int i, min, max;
    min = A[0];
    max = A[0];
    for(i=1; i < n; i++) {
        if(max < A[i]) max = A[i];
        else
            if(min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

Função de Complexidade

Exemplo

- Qual a função de complexidade de maxMin?
- Depende:
 - Se o arranjo já estiver ordenado em ordem crescente:
 $f(n) = n - 1$
Só um if será ativado

```
void minMax(int A[], int n) {  
    int i, min, max;  
    min = A[0];  
    max = A[0];  
    for(i=1; i < n; i++) {  
        if(max < A[i]) max = A[i];  
        else  
            if(min > A[i]) min = A[i];  
    }  
    printf("Minimo: %i\n", min);  
    printf("Maximo: %i\n", max);  
}
```

Função de Complexidade

Exemplo

- Qual a função de complexidade de maxMin?
- Depende:
 - Se o arranjo já estiver ordenado em ordem decrescente:

```
void minMax(int A[], int n) {
    int i, min, max;
    min = A[0];
    max = A[0];
    for(i=1; i < n; i++) {
        if(max < A[i]) max = A[i];
        else
            if(min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

Função de Complexidade

Exemplo

- Qual a função de complexidade de maxMin?
- Depende:
 - Se o arranjo já estiver ordenado em ordem decrescente:

$$f(n) = 2(n - 1)$$

```
void minMax(int A[], int n) {
    int i, min, max;
    min = A[0];
    max = A[0];
    for(i=1; i < n; i++) {
        if(max < A[i]) max = A[i];
        else
            if(min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

Função de Complexidade

Exemplo

- Qual a função de complexidade de maxMin?
- Depende:
 - Se o arranjo já estiver ordenado em ordem decrescente:
 $f(n) = 2(n - 1)$
Ambos ifs são ativados

```
void minMax(int A[], int n) {
    int i, min, max;
    min = A[0];
    max = A[0];
    for(i=1; i < n; i++) {
        if(max < A[i]) max = A[i];
        else
            if(min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

Função de Complexidade

Exemplo

```
void maxMin(int A[], int n) {
    int i, max, min;
    max = min = A[0];
    for(i=1; i < n; ++i) {
        if(max < A[i]) max = A[i];
        else
            if (min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

- Se $A[i] > \text{max}$ metade das vezes

Exemplo

```
void maxMin(int A[], int n) {
    int i, max, min;
    max = min = A[0];
    for(i=1; i < n; ++i) {
        if(max < A[i]) max = A[i];
        else
            if (min > A[i]) min = A[i];
    }
    printf("Minimo: %i\n", min);
    printf("Maximo: %i\n", max);
}
```

- Se $A[i] > \max$ metade das vezes

$$f(n) = \frac{(n-1) + 2(n-1)}{2}$$
$$= 3\frac{n}{2} - \frac{3}{2}, \text{ para } n > 0$$

Três cenários podem ser observados

- Melhor caso: já ordenado crescentemente
 - Menor tempo de execução dentre as entradas de tamanho n

Três cenários podem ser observados

- Melhor caso: já ordenado crescentemente
 - Menor tempo de execução dentre as entradas de tamanho n
- Pior caso: já ordenado decrescentemente
 - Maior tempo de execução dentre as entradas de tamanho n

Três cenários podem ser observados

- Melhor caso: já ordenado crescentemente
 - Menor tempo de execução dentre as entradas de tamanho n
- Pior caso: já ordenado decrescentemente
 - Maior tempo de execução dentre as entradas de tamanho n
- Caso médio: **não é, necessariamente, a média do melhor e do pior caso!**
 - Média dos tempos de execução de todas as entradas de tamanho n

Caso Médio

- Supõe uma distribuição de probabilidades sobre o conjunto de entradas de tamanho n
- O custo médio é obtido com base nessa distribuição.

Caso Médio

- Supõe uma distribuição de probabilidades sobre o conjunto de entradas de tamanho n
 - O custo médio é obtido com base nessa distribuição.
- Normalmente, o caso médio é muito mais difícil de determinar do que o melhor e o pior caso
 - Comumente, supõe-se que todas as entradas têm a mesma chance de ocorrer \rightarrow equiprováveis.
 - Nem sempre isto é verdade. Por isso, o caso médio é determinado apenas se fizer sentido.

Caso Médio – Exemplo

- Busca sequencial em um arranjo de tamanho n
- Qual o melhor caso?

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0; i<n; i++)
        if (A[i] == x)
            return i;

    return -1;
}
```

Caso Médio – Exemplo

- Busca sequencial em um arranjo de tamanho n
- Qual o melhor caso?
 - O valor procurado é o primeiro \rightarrow 1 comparação

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0; i<n; i++)
        if (A[i] == x)
            return i;

    return -1;
}
```

Caso Médio – Exemplo

- Busca sequencial em um arranjo de tamanho n
- Qual o melhor caso?
 - O valor procurado é o primeiro \rightarrow 1 comparação
 - $f(n) = 1$

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0; i<n; i++)
        if (A[i] == x)
            return i;
    return -1;
}
```

Caso Médio – Exemplo

- Qual o pior caso?

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0; i<n; i++)
        if (A[i] == x)
            return i;

    return -1;
}
```

Caso Médio – Exemplo

- Qual o pior caso?
 - O valor procurado é o último ou não está em A

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0; i<n; i++)
        if (A[i] == x)
            return i;

    return -1;
}
```

Caso Médio – Exemplo

- Qual o pior caso?
 - O valor procurado é o último ou não está em A
 - $f(n) = n$

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0; i<n; i++)
        if (A[i] == x)
            return i;

    return -1;
}
```

Caso Médio – Exemplo

- Melhor caso:

- $f(n) = 1$

- Pior caso:

- $f(n) = n$

- Caso médio*:

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0;i<n;i++)
        if (A[i] == x)
            return i;

    return -1;
}
```

Caso Médio – Exemplo

- Melhor caso:

- $f(n) = 1$

- Pior caso:

- $f(n) = n$

- Caso médio*:

- $f(n) = \frac{1+n}{2}$

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0;i<n;i++)
        if (A[i] == x)
            return i;

    return -1;
}
```

Caso Médio – Exemplo

- Melhor caso:

- $f(n) = 1$

- Pior caso:

- $f(n) = n$

- Caso médio*:

- $f(n) = \frac{1+n}{2}$

- *Mas isso supondo uma distribuição uniforme de casos (e elemento encontrado)...

```
int busca(int x,int A[],
          int n){
    int i;
    for (i=0;i<n;i++)
        if (A[i] == x)
            return i;
    return -1;
}
```

Função de Complexidade

Cálculo da função de complexidade:

Função de Complexidade

Cálculo da função de complexidade:

- Identifique o tempo de execução de cada comando no algoritmo

Função de Complexidade

Cálculo da função de complexidade:

- Identifique o tempo de execução de cada comando no algoritmo
- Selecione apenas os comandos relacionados com o tamanho da entrada
 - Ex: aqueles que irão ser executados um número de vezes proporcional a esse tamanho

Função de Complexidade

Cálculo da função de complexidade:

- Identifique o tempo de execução de cada comando no algoritmo
- Selecione apenas os comandos relacionados com o tamanho da entrada
 - Ex: aqueles que irão ser executados um número de vezes proporcional a esse tamanho
- A soma dos tempos de execução de cada um desses comandos corresponde ao tempo de execução do algoritmo

Função de Complexidade

Exemplo

- Determine a função de complexidade do algoritmo de ordenação por inserção no pior caso para um arranjo de tamanho n

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];  
        j = i;  
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  
            j--;  
        }  
        v[j] = aux;  
    }  
}
```

Exemplo

- Qual o pior caso?

```
void insercao(int v[], int n) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

Função de Complexidade

Exemplo

- Qual o pior caso?
- O caso em que todo laço será executado até o fim

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];  
        j = i;  
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  
            j--;  
        }  
        v[j] = aux;  
    }  
}
```

Função de Complexidade

Exemplo

- Qual o pior caso?
 - O caso em que todo laço será executado até o fim
 - Ex: v ordenado em ordem inversa ou desordenado de modo a que essa condição ocorra

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];  
        j = i;  
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  
            j--;  
        }  
        v[j] = aux;  
    }  
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

$n - 1$

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];  
        j = i;  
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  
            j--;  
        }  
        v[j] = aux;  
    }  
}
```

$n - 1$

O laço executa
 $n - 1$ repetições
desse comando

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];           n - 1
        j = i;             n - 1
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];           n - 1
        j = i;               n - 1
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];   ???
            j--;
        }
        v[j] = aux;
    }
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

- Vejamos as iterações do while mais de perto

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

- Vejamos as iterações do while mais de perto
- No pior caso, essa parte não nos ajudará

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

i	iterações
1	

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

i	iterações
1	1

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

i	iterações
1	1
2	

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

i	iterações
1	1
2	2

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

i	iterações
1	1
2	2
...	...

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

i	iterações
1	1
2	2
...	...
$n - 1$	

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
int i, j, aux;
for (i=1; i<n; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
           (aux < v[j-1])) {
        v[j] = v[j-1];
        j--;
    }
    v[j] = aux;
}
```

i	iterações
1	1
2	2
...	...
$n - 1$	$n - 1$

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
for (i=1; i<n; i++)  
{  
    aux = v[i];  
    j = i;  
    while ((j > 0) &&  
           (aux < v[j-1])) {  
        v[j] = v[j-1];  
        j--;  
    }  
    v[j] = aux;  
}
```

Então, temos

$$1 + 2 + \dots + (n - 1) \\ = \frac{n(n - 1)}{2}$$

repetições dos
comandos no while

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];            $n - 1$   
        j = i;                 $n - 1$   
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  
            j--;  
        }  
        v[j] = aux;  
    }  
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];  $n - 1$   
        j = i;  $n - 1$   
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  $\frac{n(n-1)}{2}$   
            j--;  
        }  
        v[j] = aux;  
    }  
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];  $n - 1$   
        j = i;  $n - 1$   
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  $\frac{n(n-1)}{2}$   
            j--;  $\frac{n(n-1)}{2}$   
        }  
        v[j] = aux;  
    }  
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int v[], int n) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i]; n - 1  
        j = i; n - 1  
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  $\frac{n(n-1)}{2}$   
            j--;  $\frac{n(n-1)}{2}$   
        }  
        v[j] = aux; n - 1  
    }  
}
```

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int[] v) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

- E juntando tudo teremos:^a

$$3(n-1) + 2\frac{n(n-1)}{2}$$

^aObserve que havia outras operações (mas iremos ignorá-las, por enquanto).

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int[] v) {  
    int i, j, aux;  
    for (i=1; i<n; i++) {  
        aux = v[i];  
        j = i;  
        while ((j > 0) &&  
                (aux < v[j-1])) {  
            v[j] = v[j-1];  
            j--;  
        }  
        v[j] = aux;  
    }  
}
```

- E juntando tudo teremos:^a

$$3(n-1) + 2\frac{n(n-1)}{2}$$

^aObserve que havia outras operações (mas iremos ignorá-las, por enquanto).

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int[] v) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

- E juntando tudo teremos:^a

$$3(n-1) + 2 \frac{n(n-1)}{2}$$

^aObserve que havia outras operações (mas iremos ignorá-las, por enquanto).

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int[] v) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

- E juntando tudo teremos:^a

$$3(n-1) + 2 \frac{n(n-1)}{2} \\ = 3(n-1) + n(n-1)$$

^aObserve que havia outras operações (mas iremos ignorá-las, por enquanto).

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int[] v) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

- E juntando tudo teremos:^a

$$\begin{aligned} & 3(n-1) + 2\frac{n(n-1)}{2} \\ &= 3(n-1) + n(n-1) \\ &= 3n - 3 + n^2 - n \end{aligned}$$

^aObserve que havia outras operações (mas iremos ignorá-las, por enquanto).

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

```
void insercao(int[] v) {
    int i, j, aux;
    for (i=1; i<n; i++) {
        aux = v[i];
        j = i;
        while ((j > 0) &&
                (aux < v[j-1])) {
            v[j] = v[j-1];
            j--;
        }
        v[j] = aux;
    }
}
```

- E juntando tudo teremos:^a

$$\begin{aligned} & 3(n-1) + 2\frac{n(n-1)}{2} \\ &= 3(n-1) + n(n-1) \\ &= 3n - 3 + n^2 - n \\ &= n^2 + 2n - 3 \end{aligned}$$

^aObserve que havia outras operações (mas iremos ignorá-las, por enquanto).

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

- Naturalmente, os coeficientes de $f(n) = n^2 + 2n - 3$ dependem de quais operações (e como) contamos
- Neste caso:

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

- Naturalmente, os coeficientes de $f(n) = n^2 + 2n - 3$ dependem de quais operações (e como) contamos
- Neste caso:
 - Contamos $\text{aux} = v[i]$ e $j = i$, não apenas comparações (como $\text{aux} < v[j-1]$, por exemplo)

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

- Naturalmente, os coeficientes de $f(n) = n^2 + 2n - 3$ dependem de quais operações (e como) contamos
- Neste caso:
 - Contamos $\text{aux} = v[i]$ e $j = i$, não apenas comparações (como $\text{aux} < v[j-1]$, por exemplo)
- Nada disso importa

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

- Naturalmente, os coeficientes de $f(n) = n^2 + 2n - 3$ dependem de quais operações (e como) contamos
- Neste caso:
 - Contamos $\text{aux} = v[i]$ e $j = i$, não apenas comparações (como $\text{aux} < v[j-1]$, por exemplo)
- Nada disso importa
 - O que nos importa é que, no todo, nosso algoritmo varia quadraticamente (n^2) com o tamanho da entrada

Função de Complexidade

Exemplo – Contemos as operações $|v| = n$

- Naturalmente, os coeficientes de $f(n) = n^2 + 2n - 3$ dependem de quais operações (e como) contamos
- Neste caso:
 - Contamos $\text{aux} = v[i]$ e $j = i$, não apenas comparações (como $\text{aux} < v[j-1]$, por exemplo)
- Nada disso importa
 - O que nos importa é que, no todo, nosso algoritmo varia quadraticamente (n^2) com o tamanho da entrada
 - Os coeficientes variarão conforme a implementação, mas não seu comportamento quadrático

Função de Complexidade

Em suma...

- Problemas requerem algoritmos que os solucionem
- O algoritmo adequado depende do seu comportamento
 - Complexidade temporal e espacial
- Algoritmo ótimo
 - Soluciona o problema com o menor custo possível
- Função de complexidade
 - Melhor caso, pior caso e caso médio

Referências

- Ziviani, Nivio. Projeto de Algoritmos: com implementações em Java e C++. Cengage. 2007.
- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.
- Gersting, Judith L. Fundamentos Matemáticos para a Ciência da Computação. 3a ed. LTC. 1993.

Aula 02 – Custo de um Algoritmo e Complexidade

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023