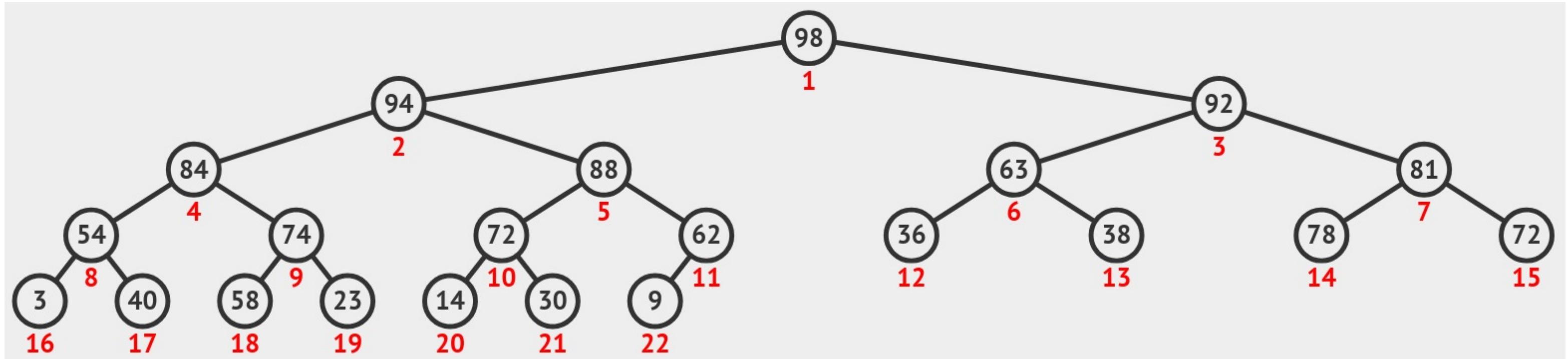
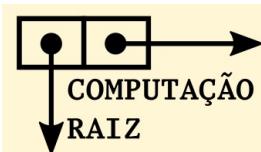


Estruturas de Dados e TADs



Tipos de Dados x Estruturas de Dados x Tipos Abstratos de Dados (TAD)

- Uma **variável** é uma estrutura de dados?
- Um **tipo de dado** é uma estrutura de dados?
- Um **array** é uma estrutura de dados?
- Um **tipo abstrato de dado** é uma estrutura de dados?
- Uma **pilha** é uma estrutura de dados?
- Um **dicionário** é uma estrutura de dados? Ou um TAD?
- Um **heap binário** é uma estrutura de dados? Um tipo de dado? Ou um TAD?
- Uma estrutura de dados tem **interface**?
- Uma **árvore** é um tipo de dado, um TAD ou uma estrutura de dados?
- O que é uma estrutura de dados **híbrida**?
- Qual a diferença entre:
 - **Tipo de dado**
 - **Tipo abstrato de dado (TAD)**
 - **Tipo concreto de dado (TCD)**
 - **Estruturas de dados**
- ...



Confusão enorme!

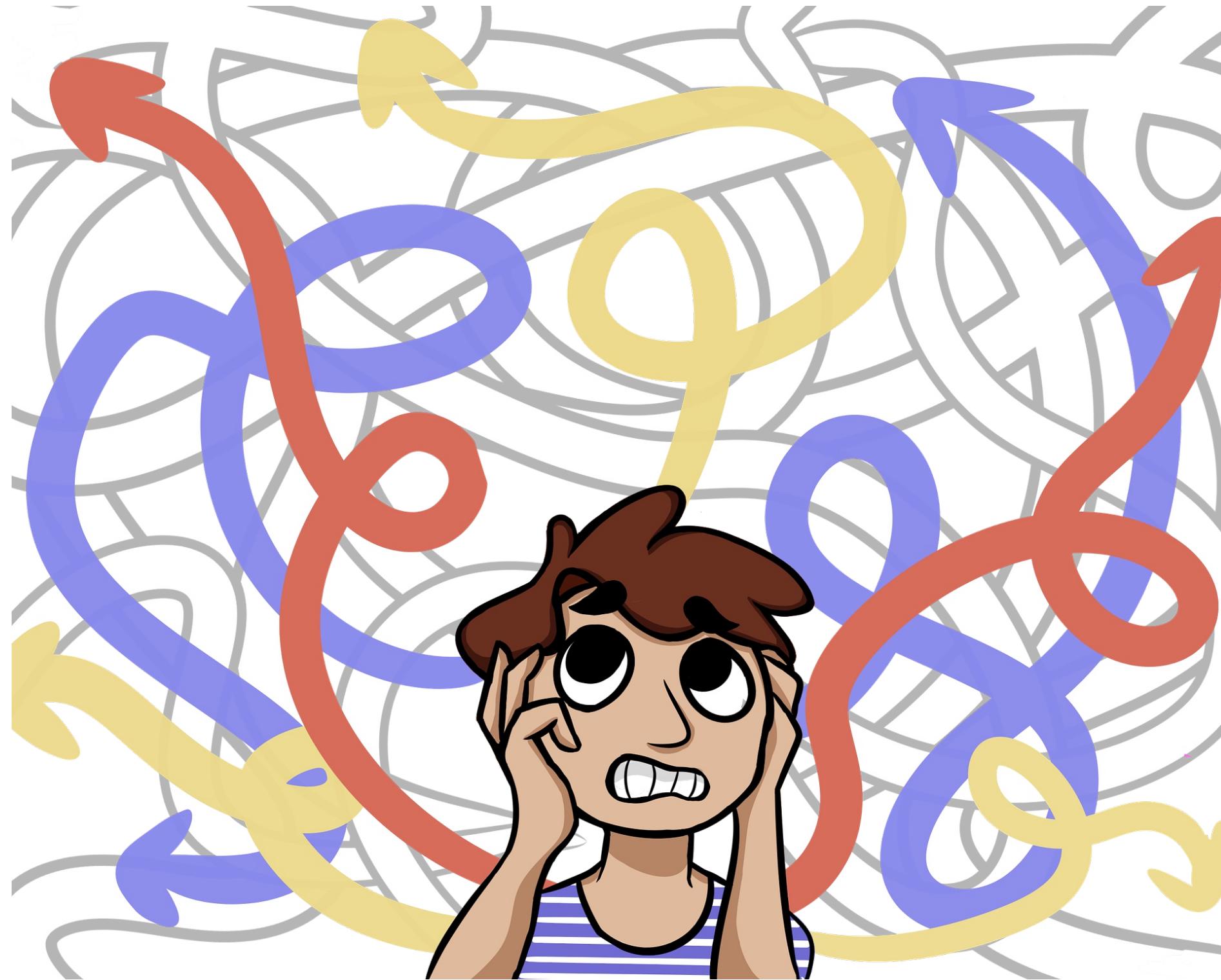
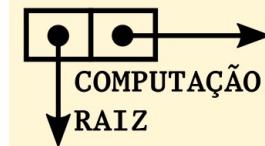
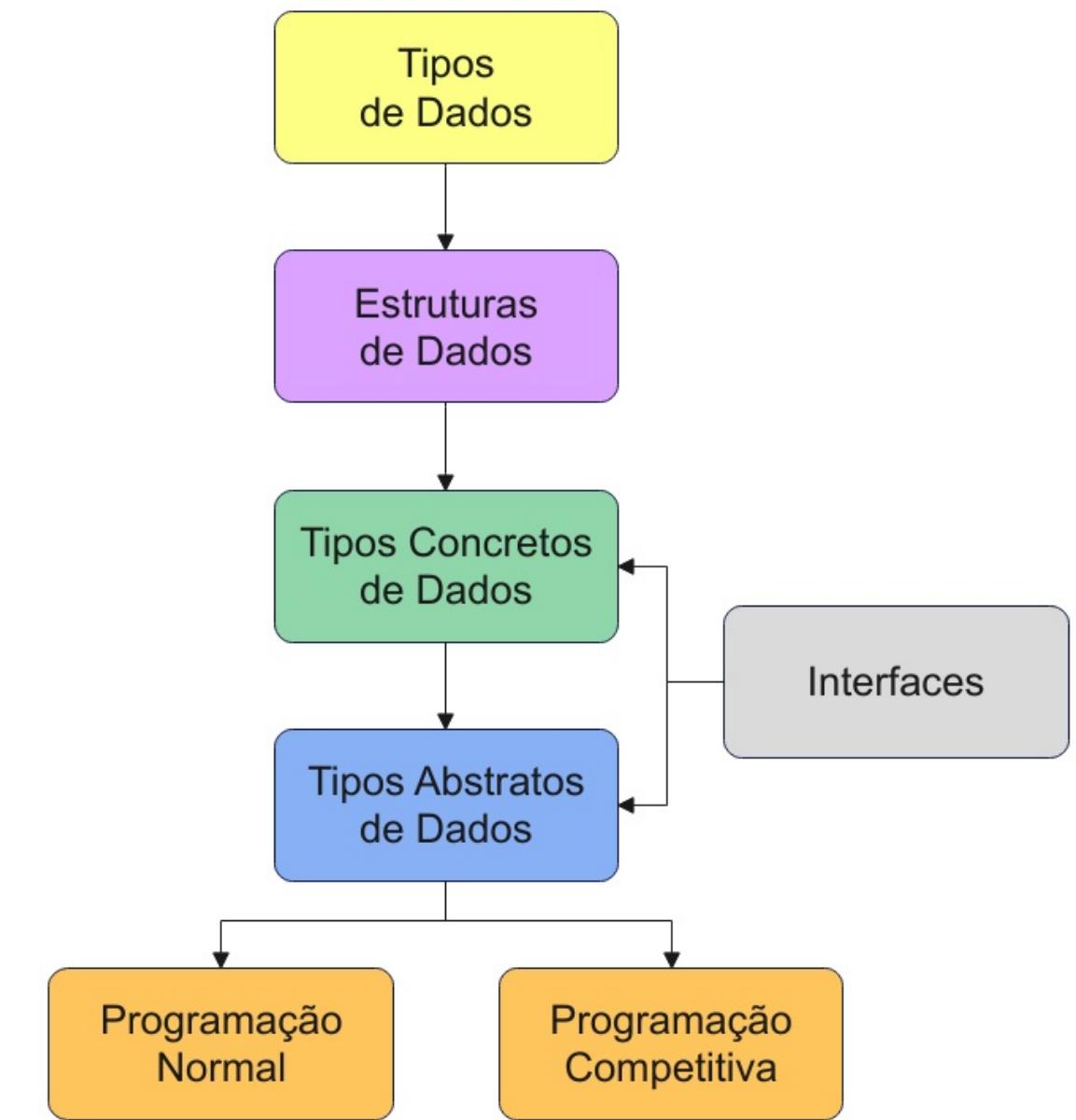
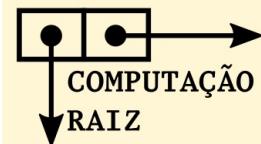
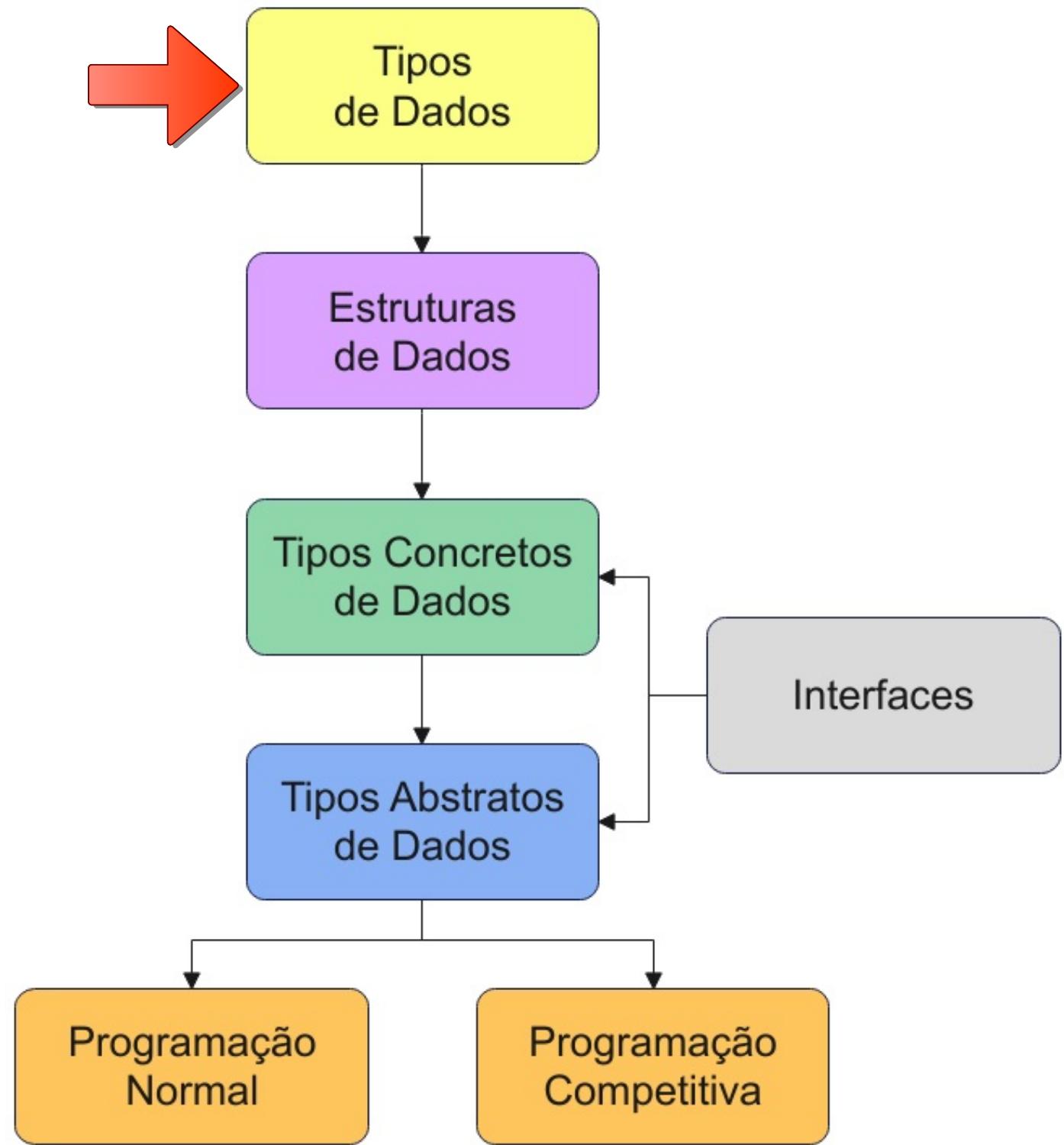


Imagen: MMillustrates, no Pixabay (<https://pixabay.com/illustrations/unordered-chaos-3192273/>)



Tipos de Dados

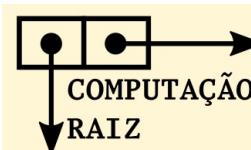


Tipos de Dados

Ao final de tudo, os **dados** que usamos em nossos programas **são apenas bits 0 e 1**. Em geral esses dados derivam de descrições **matemáticas** ou da **linguagem natural**, mas se nossos programas tivessem que processar diretamente esses bits para interpretar os **números e caracteres**, a programação seria difícil.

É necessário uma **abstração**, alguma coisa que nos permita trabalhar com conjuntos e padrões particulares de bits para podermos entender e processar os bits em um **nível mais alto**.

A abstração necessária para isso é o **Tipo de Dado.**



Tipos de Dados

Um Tipo de Dado é uma maneira de especificar como determinados padrões particulares de bits 0s e 1s serão interpretados e manipulados para representar números e caracteres (e outros tipos de informações).

Considere este padrão: 01000001

- Que valor está representado?

Depende do tipo de dado!

- Se for inteiro unsigned: 65
 - Se for caractere: A

- Que manipulações (operações) podem ser feitas?

Depende do tipo de dado!

- Aritméticas
 - Operações de caracteres e strings



Tipo de dado = Conjunto de Valores + Conjunto de Operações



Tipos de Dados

Idealmente só precisaríamos de 2 tipos de dados:

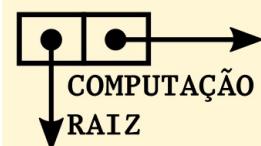
- números
- caracteres

Na prática, como a **memória é limitada**, criamos mais tipos de dados do que o necessário (**trocamos espaço por exatidão/precisão**):

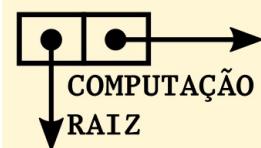
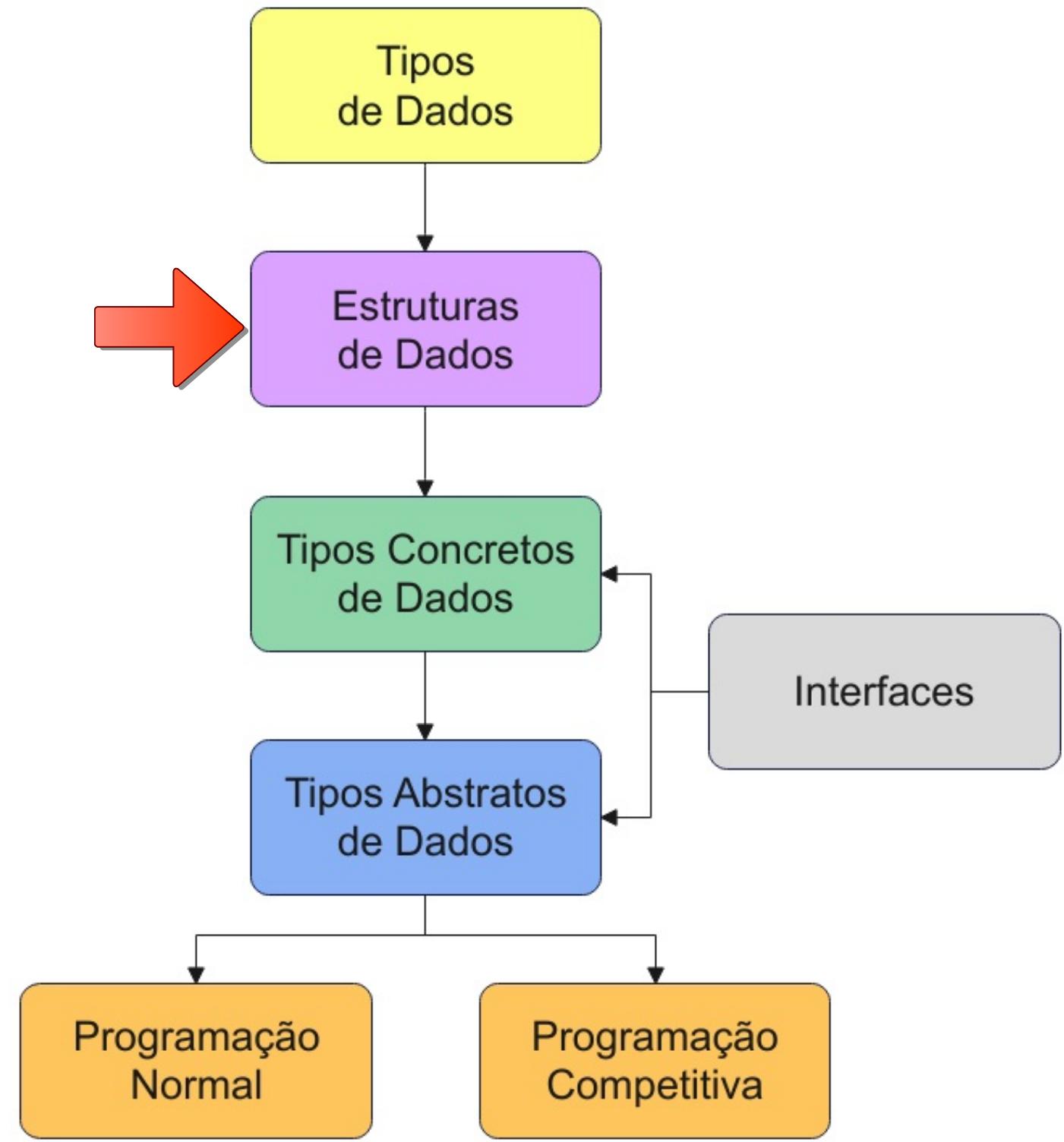
[unsigned] short int	float
[unsigned] int	double
[unsigned] long int	long double
[unsigned] long long int	[unsigned/signed] char

Além disso, o número de bits que são utilizados para representar cada um desses tipos de dados é **variável por arquitetura e sistema operacional**. Isso é fonte freqüente de erros! Em geral:

short int	(16 bits)	float	(32 bits)
int	(32 bits)	double	(64 bits)
long int	(64 bits)	long double	(80 bits)
long long int	(64 bits)	char	(8 bits)



Estruturas de Dados



Estruturas de Dados

São estruturas de memória que armazenam os dados (valores) que estão manipulados por um algoritmo, de forma organizada e otimizada, para aumentar a eficiência do processamento que está sendo realizado.

O mesmo dado pode ser armazenado por diferentes estruturas e, por isso, a escolha de qual estrutura de dados utilizar é fundamental:

- Podem ocupar mais ou menos espaço
- Podem permitir algoritmos mais eficientes (tempo/espaço)
- Simplificam os algoritmos
- Simplificam o programa como um todo

É mais fácil mudar a estrutura de dados por uma mais eficiente do que criar um algoritmo mais complexo para uma estrutura não adequada para o problema em questão.

9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0
<hr/>									
1	1	0	1	0	1	1	0	1	0

= indices
= S = 859
= mask
= Result

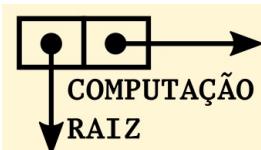


Estruturas de Dados

Que estruturas de dados existem? VÁRIAS, MUITAS MESMO!

Vamos estudá-las, didaticamente, usando a seguinte divisão:

- 1) Estruturas que armazenam **dados que se comportam como unidade**
- 2) Estruturas que armazenam **diversos dados ao mesmo tempo**

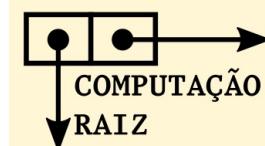


Estruturas de Dados

1) Estruturas de dados para armazenar **dados que se comportam como uma unidade**:

O que são **dados que se comportam como uma unidade**?

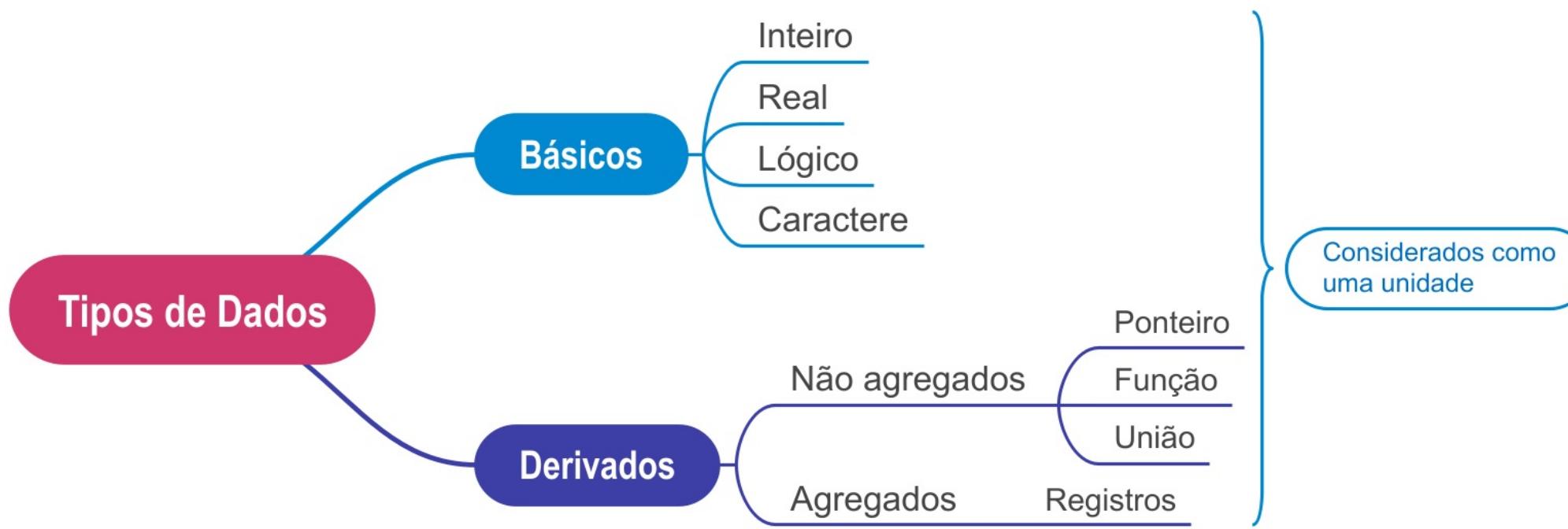
- **Básicos** (primitivos): não podem ser decompostos em partes menores, representam o entendimento direto dos padrões de bits como dados.
- **Derivados**: são construídos a partir dos tipos de dados básicos mas ainda são considerados como uma "unidade".



Estruturas de Dados

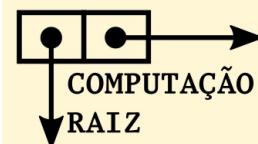
1) Estruturas de dados para armazenar **dados que se comportam como uma unidade**:

Que estrutura de dados podemos utilizar? Como os dados são considerados como uma "coisa única", basta uma simples **variável**!



Estrutura de Dados

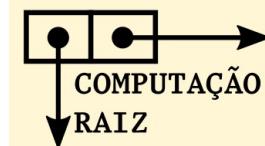
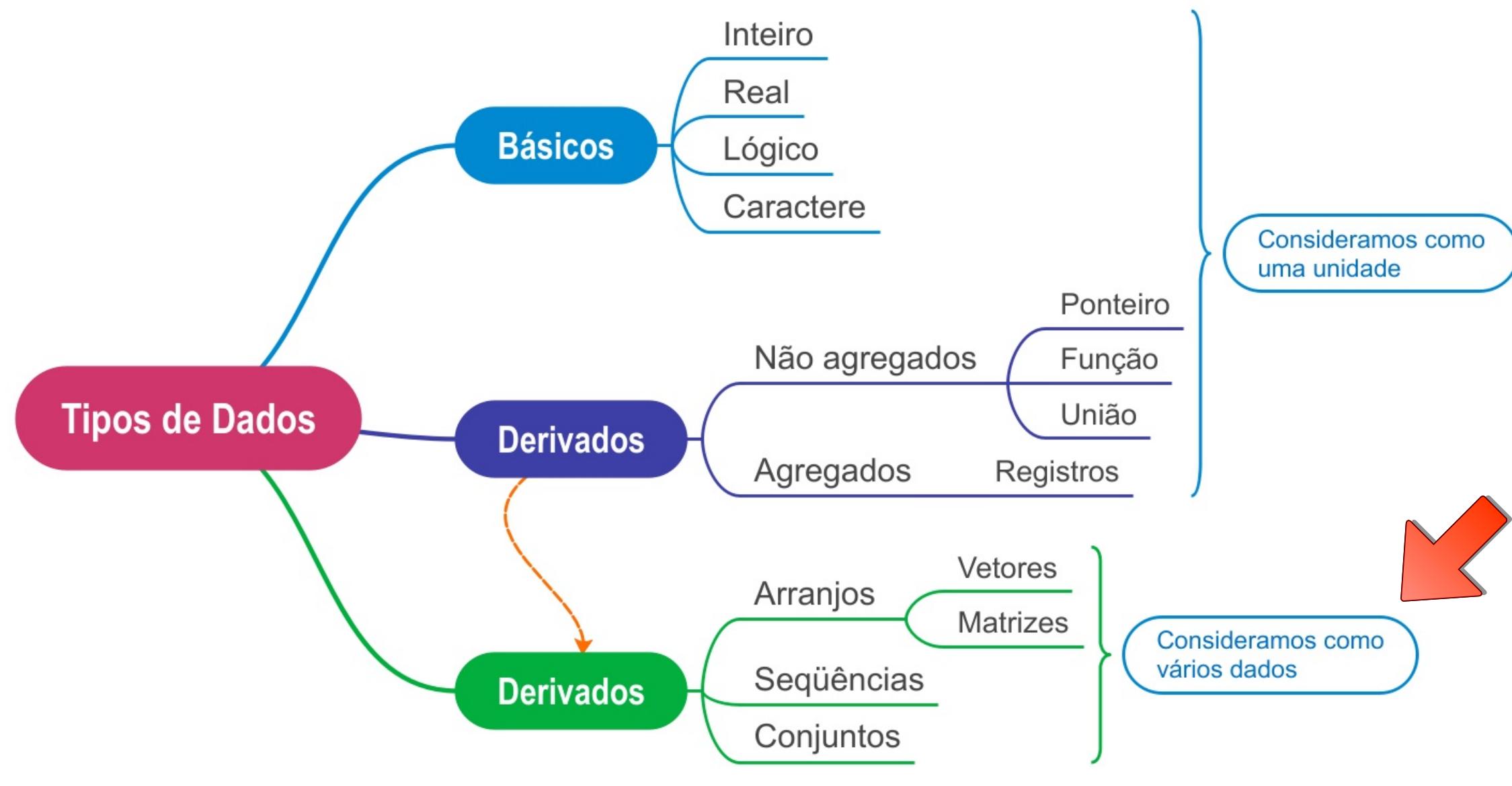
Variável



Estruturas de Dados

2) Estruturas de dados para armazenar **diversos dados ao mesmo tempo**

Agora precisamos de algo para armazenar diversos valores ao mesmo tempo, o que não pode ser feito com uma única variável.



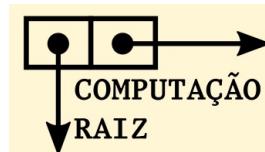
Estruturas de Dados

2) Estruturas de dados para armazenar **diversos dados ao mesmo tempo**

Que estrutura de dados podemos utilizar? Aqui a coisa é mais complicada pois como temos muitos dados precisamos verificar qual o **relacionamento** entre esses dados.

Temos que verificar:

- **Ordenamento**: a **ordenação** dos dados importa?
- **Linearidade**: os dados formam uma estrutura **linear** ou **não linear**?
 - Se são **não lineares**, há **hierarquia** ou são **não hierárquicos**?
- **Outros**



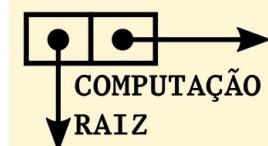
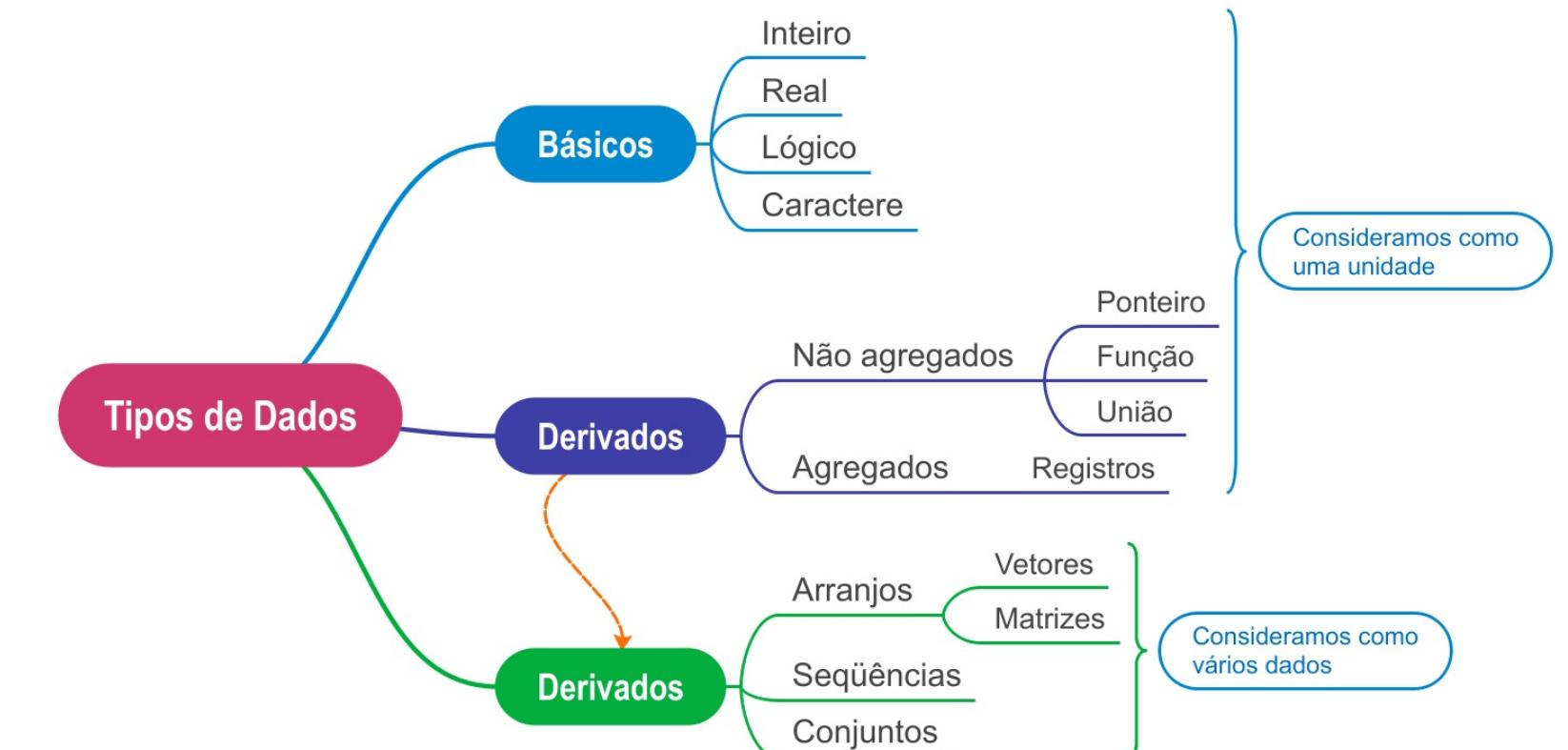
Estruturas de Dados

2) Estruturas de dados para armazenar **diversos dados ao mesmo tempo**

Para representar esses dados (ordenados ou não, lineares ou não lineares), temos 3 grandes grupos de estruturas:

Por contigüidade
Por encadeamento
Híbridas

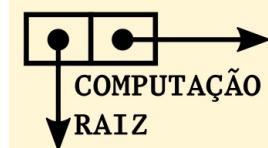
- estruturas **lineares**
- estruturas **não lineares**
 - hierárquicas
 - não hierárquicas
- outras



Estruturas de Dados

2) Estruturas de dados para armazenar **diversos dados ao mesmo tempo**

- **Contigüidade**: os dados são armazenados em posições contíguas na memória. A ordem é definida implicitamente pela posição, geralmente com tamanho máximo definido previamente.
- **Encadeamento**: os dados são armazenados em posições aleatórias na memória alocadas dinamicamente, com ponteiros para indicar o endereço do próximo dado.
- **Híbridas**: misturam dados contíguos com encadeados.

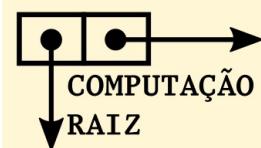


Estruturas de Dados

2) Estruturas de dados para armazenar **diversos dados ao mesmo tempo**



Algumas estruturas podem ser implementadas de várias formas.

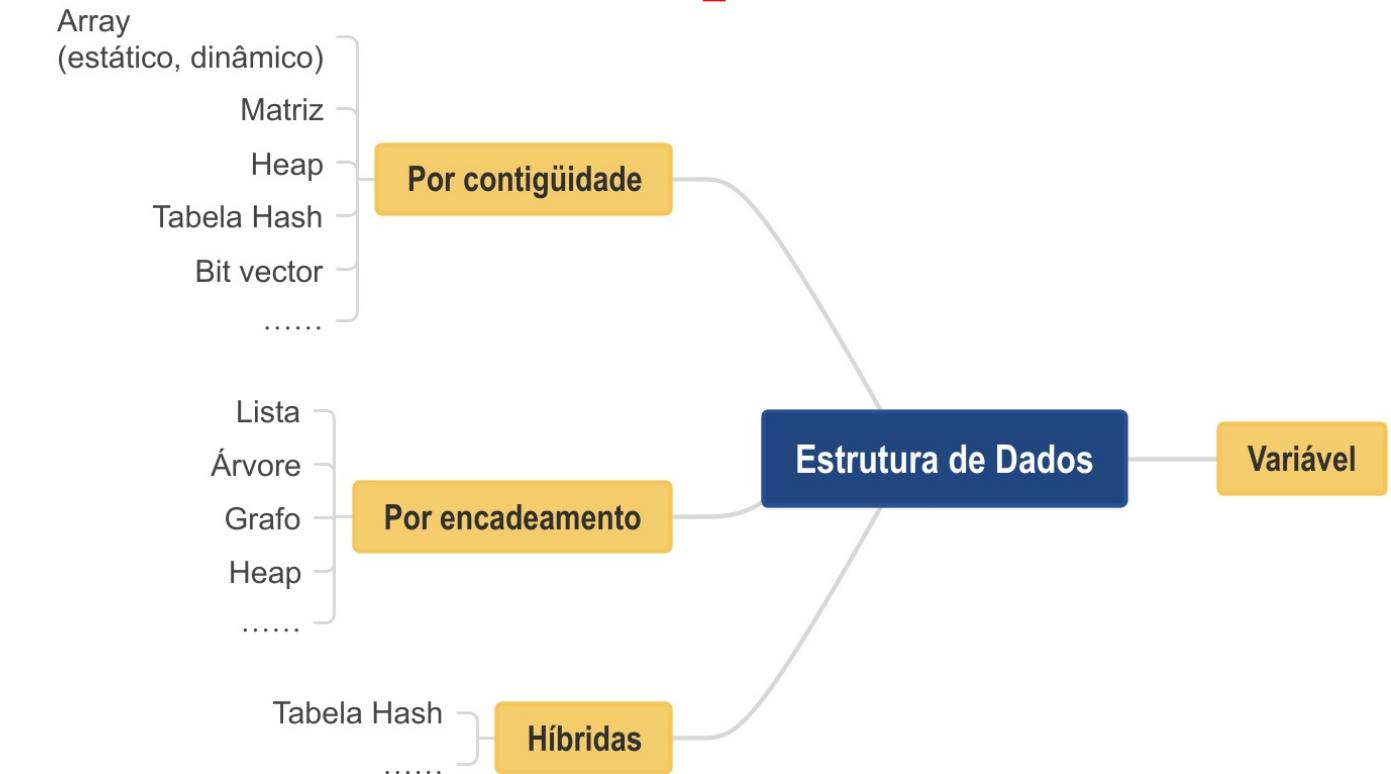


Estruturas de Dados

2) Estruturas de dados para armazenar **diversos dados ao mesmo tempo**

Estruturas por Contigüidade:

- **Vantagens:**
 - Tempo de acesso
 - Eficiência de espaço (só dados, sem ponteiros)
 - Localidade de memória (transferência, iteração, cache)
- **Desvantagens:**
 - Não ajusta tamanho em tempo de execução (exceto array dinâmico)
 - Não permite compartilhamento de memória
 - Inserção e exclusão complexa e custosa (deslocamento de dados)

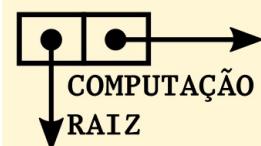
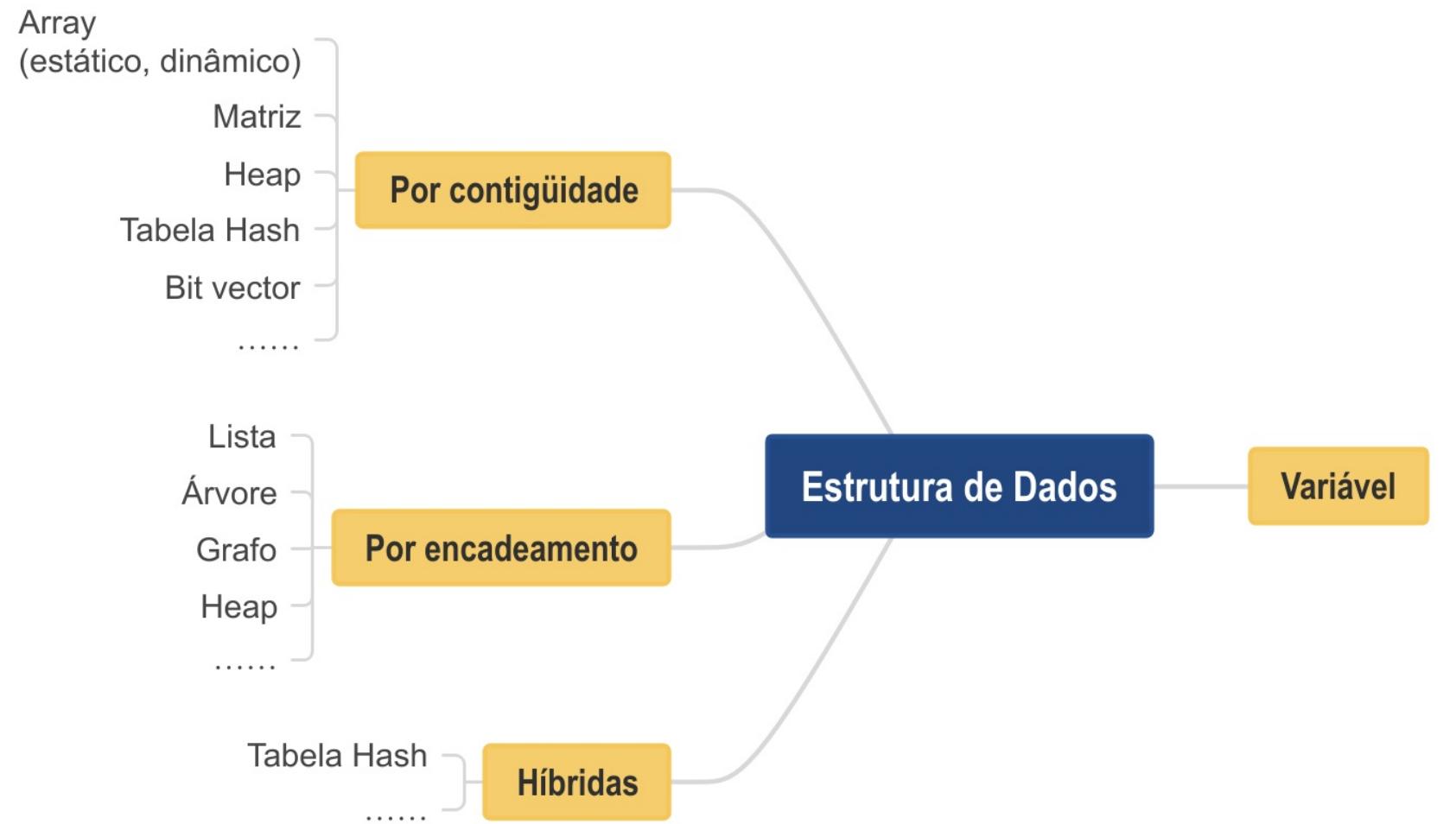


Estruturas de Dados

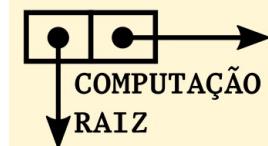
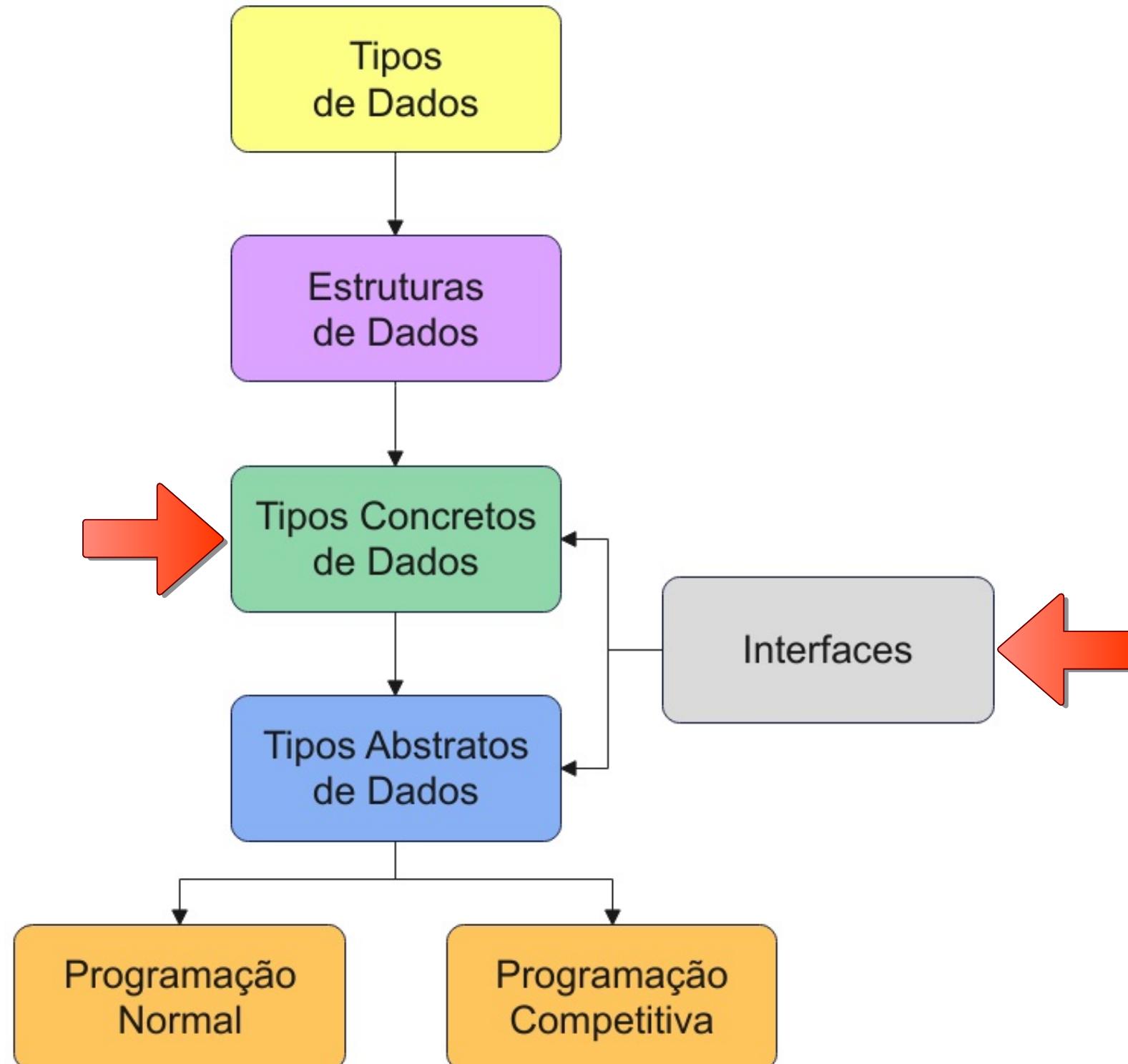
2) Estruturas de dados para armazenar **diversos dados ao mesmo tempo**

Estruturas por Encadeamento:

- **Vantagens:**
 - Compartilhamento de memória
 - Flexibilidade
 - Inclusão e exclusão facilitada
- **Desvantagens:**
 - Não localidade de memória (transferência, iteração e cache)
 - Menor eficiência de espaço (dados e ponteiros)
 - Acesso e busca complexos e custosos

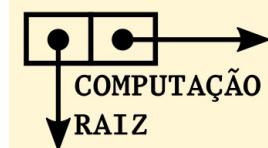
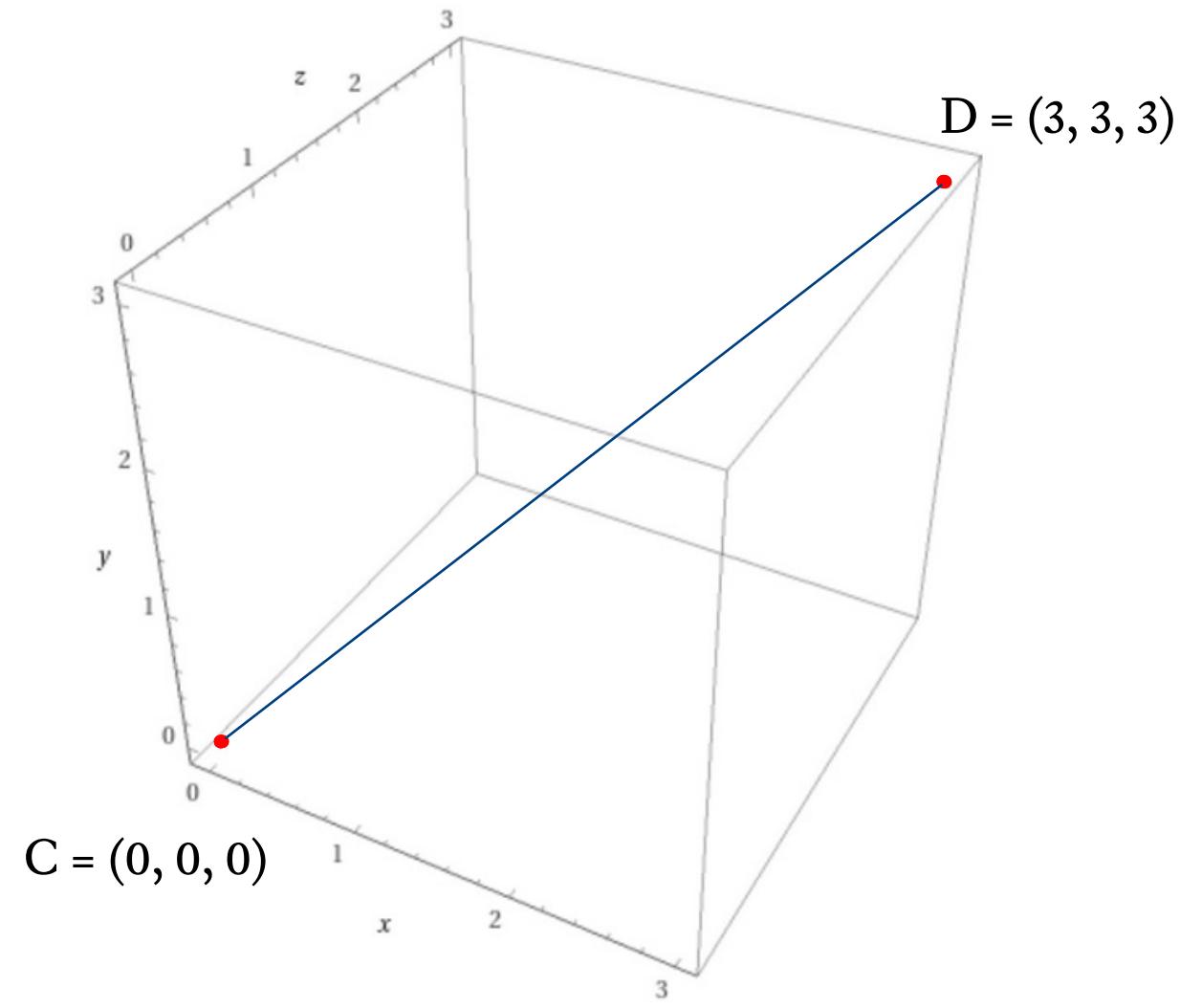
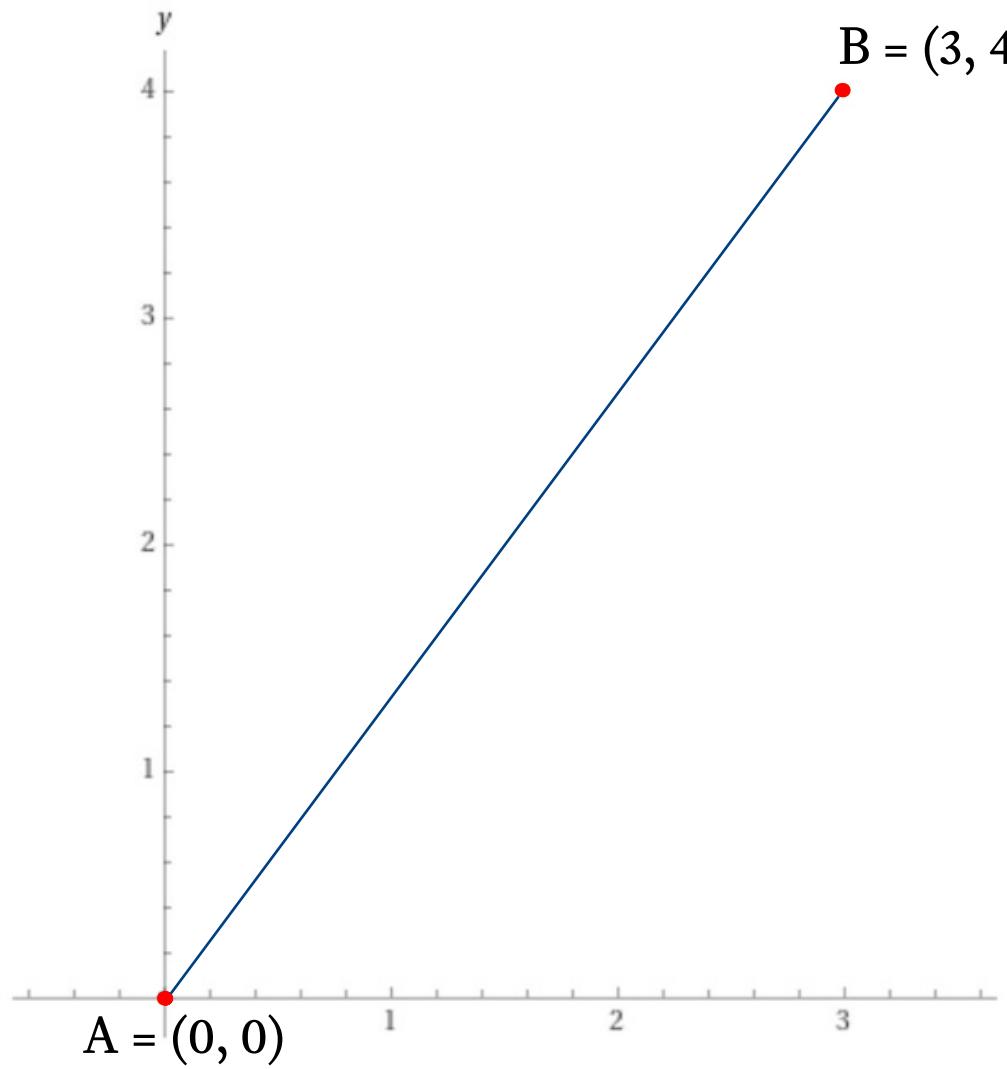


Tipos Concretos de Dados



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

Em diversas situações precisamos criar nossos próprios tipos de dados. Isso é feito **combinando-se** as **estruturas de dados** e especificando os **comportamentos (operações)**. Vamos criar tipos para pontos no plano e no espaço cartesiano, para calcular as distâncias:



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

```
1 #ifndef PONTOS_H
2 #define PONTOS_H
3
4 /** Estruturas */
5
6 struct st_Ponto2D
7 {
8     double x, y;
9 };
10
11 typedef struct st_Ponto2D Ponto2D;
12
13 struct st_Ponto3D
14 {
15     double x, y, z;
16 };
17
18 typedef struct st_Ponto3D Ponto3D;
19
20 /** Comportamentos */
21
22 double euclidiana_2d (Ponto2D P, Ponto2D Q);
23
24 double euclidiana_3d (Ponto3D P, Ponto3D Q);
25
26#endif
```

Tipo de dado = Conjunto de Valores + Conjunto de Operações



As *headers files*, em C e C++, foram a interface de uma biblioteca de tipos de dados, com valores e comportamentos (operações).



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

```
1 #include "pontoTCD.h"
2 #include <math.h>
3
4 /** Comportamentos */
5
6 double euclidiana_2d (Ponto2D P, Ponto2D Q)
7 {
8     return sqrt(pow((P.x - Q.x), 2.0) +
9                 pow((P.y - Q.y), 2.0));
10 }
11
12 double euclidiana_3d (Ponto3D P, Ponto3D Q)
13 {
14     return sqrt(pow((P.x - Q.x), 2.0) +
15                 pow((P.y - Q.y), 2.0) +
16                 pow((P.z - Q.z), 2.0));
17 }
```



pontoTCD.c

Além das *headers files*, em C e C++, temos que ter a **implementação da interface**, que realmente implementa os comportamentos (operações). Note que a implementação inclui a interface.

Tipo de dado = Conjunto de Valores + Conjunto de Operações

```
gcc -std=c17 -Wall -Wpedantic -Wconversion -Werror -Wunused-result -c -o pontoTCD.o pontoTCD.c
```



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

```
1 #include "pontoTCD.h"
2 #include <stdio.h>
3
4 int main (void)
5 {
6     Ponto2D A, B;
7     A.x = 0, A.y = 0, B.x = 3, B.y = 4;
8
9     double dist_A_B = euclidiana_2d(A, B);
10    printf("A dist. entre A=(%.2f, %.2f) e B=(%.2f, %.2f) ", 
11           A.x, A.y, B.x, B.y);
12    printf("é de %.2f.\n", dist_A_B);
13
14    Ponto3D C, D;
15    C.x = 0, C.y = 0, C.z = 0, D.x = 3, D.y = 3, D.z = 3;
16
17    double dist_C_D = euclidiana_3d(C, D);
18    printf("A dist. entre C=(%.2f, %.2f, %.2f) e D=(%.2f, %.2f, %.2f) ", 
19           C.x, C.y, C.z, D.x, D.y, D.z);
20    printf("é de %.2f.\n", dist_C_D);
21 }
```

```
gcc -std=c17 -Wall -Wpedantic -Wconversion -Werror -Wunused-result -o cliente1 cliente1.c pontoTCD.o -lm
```

```
./cliente1
A dist. entre A=(0.00, 0.00) e B=(3.00, 4.00) é de 5.00.
A dist. entre C=(0.00, 0.00, 0.00) e D=(3.00, 3.00, 3.00) é de 5.20.
```



O programa cliente inclui a interface e tem acesso ao novo tipo de dado.



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

Algumas questões importantes sobre nossos tipos de dados (Ponto2D e Ponto3D):

- 1) Eles são realmente tipos de dados? Por quê?**
- 2) A interface está bem projetada?**
- 3) O comportamento (operações) dos tipos está definido?**
- 4) Há algum problema com esses tipos de dados?**



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

Algumas questões importantes sobre nossos tipos de dados (Ponto2D e Ponto3D):

1) Eles são realmente tipos de dados? Por quê?

- Sim! Especificam os valores e comportamentos (operações).

2) A interface está bem projetada?

- Sim!

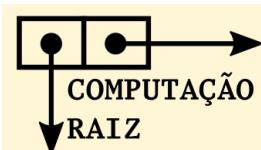
3) O comportamento (operações) dos tipos está definido?

- Sim (só tem 1 comportamento para cada tipo, mas OK)

4) Há algum problema com esses tipos de dados?

- Depende... se for só para você usar, OK.

Mas se for para terceiros usarem, há um problema grave:
o tipo de dado é **CONCRETO** e deveria ser **ABSTRATO**.



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

Em um **Tipo Concreto de Dado (TCD)**, a representação interna de como os valores são armazenados fica visível e disponível para o usuário (a estrutura de dados utilizada para guardar os dados fica visível para o usuário). Isso permite que o usuário utilize diretamente essas estruturas, e isso é um erro! Por quê?

```
6 struct st_Ponto2D
7 {
8     double x, y;
9 }
10 typedef struct st_Ponto2D Ponto2D;
11
12 struct st_Ponto3D
13 {
14     double x, y, z;
15 };
16
17 typedef struct st_Ponto3D Ponto3D;
```

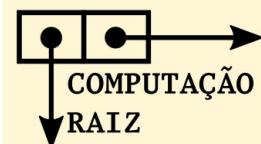
```
Ponto2D A, B;
A.x = 0, A.y = 0, B.x = 3, B.y = 4;

Ponto3D C, D;
C.x = 0, C.y = 0, C.z = 0, D.x = 3, D.y = 3, D.z = 3;
```

O cliente utiliza diretamente a estrutura de dados que implementa o tipo de dado. Isso é um erro!

pontoTCD.h

cliente1.c

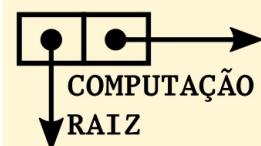


Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

Problemas com TCD: como a barreira da interface foi quebrada, o cliente pode tentar recriar os comportamentos e errar na programação ou passagem de valores:

```
1 #include <math.h>
2 #include "pontoTCD.h"
3 #include <stdio.h>
4
5 double minha_euclidiana_2d (double x1, double x2, double y1, double y2)
6 {
7     return sqrt(pow((x1 - x2), 2.0) + pow((y1 - y2), 2.0));
8 }
9
10 int main (void)
11 {
12     Ponto2D A, B;
13     A.x = 0, A.y = 0, B.x = 3, B.y = 4;
14
15     double dist_A_B = euclidiana_2d(A, B);
16     printf("A dist. entre A=(%.2f, %.2f) e B=(%.2f, %.2f) ",
17           A.x, A.y, B.x, B.y);
18     printf("é de %.2f.\n", dist_A_B);
19
20     double minha_dist = minha_euclidiana_2d(A.x, A.y, B.x, B.y);
21     printf("A dist. entre A=(%.2f, %.2f) e B=(%.2f, %.2f) ",
22           A.x, A.y, B.x, B.y);
23     printf("é de %.2f.\n", minha_dist);
24 }
```

./cliente2
A dist. entre A=(0.00, 0.00) e B=(3.00, 4.00) é de 5.00.
A dist. entre A=(0.00, 0.00) e B=(3.00, 4.00) é de 1.00.



Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

Problemas com TCD: como a barreira da interface foi quebrada, o criador do tipo de dado pode trocar a representação interna, fazendo todos os clientes quebrarem.

```
1 #ifndef PONTOS_H
2 #define PONTOS_H
3
4 /** Estruturas */
5
6 struct st_Ponto2D
7 {
8     double dados[2] = {0};
9 };
10
11 typedef struct st_Ponto2D Ponto2D;
12
13 struct st_Ponto3D
14 {
15     double dados[3] = {0};
16 };
17
18 typedef struct st_Ponto3D Ponto3D;
19
20 /** Comportamentos */
21
22 double euclidiana_2d (Ponto2D P, Ponto2D Q);
23
24 double euclidiana_3d (Ponto3D P, Ponto3D Q);
25
26#endif
```

pontoTCD.h

```
1 #include "pontoTCD.h"
2 #include <math.h>
3
4 /** Comportamentos */
5
6 double euclidiana_2d (Ponto2D P, Ponto2D Q)
7 {
8     return sqrt(pow((P.dados[0] - Q.dados[0]), 2.0) +
9                 pow((P.dados[1] - Q.dados[1]), 2.0));
10 }
11
12 double euclidiana_3d (Ponto3D P, Ponto3D Q)
13 {
14     return sqrt(pow((P.dados[0] - Q.dados[0]), 2.0) +
15                 pow((P.dados[1] - Q.dados[1]), 2.0) +
16                 pow((P.dados[2] - Q.dados[2]), 2.0));
17 }
```

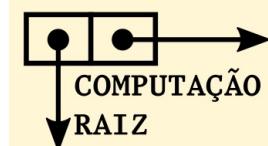
pontoTCD.c

Ponto2D A, B;
A.x = 0, A.y = 0, B.x = 3, B.y = 4;

Ponto3D C, D;
C.x = 0, C.y = 0, C.z = 0, D.x = 3, D.y = 3, D.z = 3;

Não compila mais!

cliente1.c

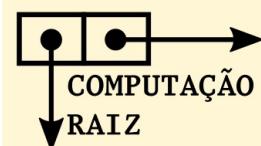


Tipos de dados criados pelo programador: Tipo Concreto de Dados (TCD)

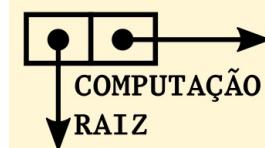
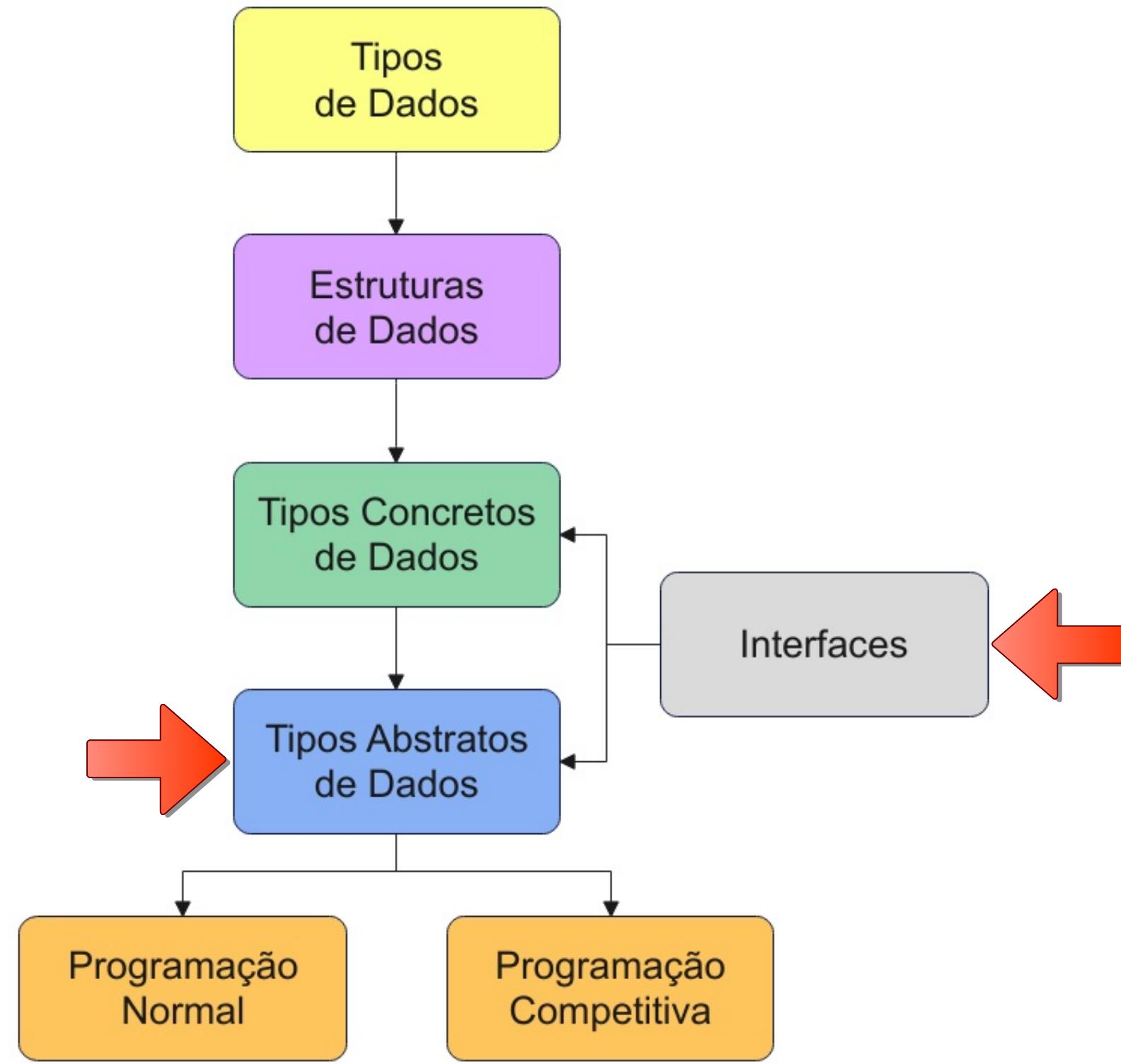
TCDs podem ser utilizados para tarefas rápidas, programas pequenos e/ou para programas que você não distribuirá para terceiros.

Se você criar um TCD e não distribuir para ninguém usar, então não há nenhum problema: você criou, você usa e você é responsável.

Mas se você está criando tipos de dados (valores e comportamentos) que outras pessoas deverão utilizar, então o TCD não é a melhor opção. Nessa situação o melhor é criar um **Tipo Abstrato de Dado (TAD)**.



Tipos Abstratos de Dados



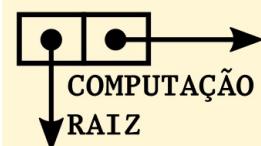
Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

Um **Tipo Abstrato de Dado (TAD)** é definido apenas em termos de seu comportamento (operações), ao invés de sua estrutura interna que fica oculta.

Em um TAD a estrutura interna não é importante para o usuário e, portanto, deve obrigatoriamente ficar oculta, opaca, escondida.

Em um TAD a interface esconde completamente a estrutura interna da implementação. O usuário não consegue, de jeito nenhum, acessar os detalhes internos da implementação e, portanto, ele é obrigado a utilizar os comportamentos (operações) da interface para tudo.

Em C/C++ isso é feito através do uso de ponteiros!



Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

```
1 #ifndef PONTOSTAD_H
2 #define PONTOSTAD_H
3
4 typedef struct st_Ponto2D *Ponto2D;
5 typedef struct st_Ponto3D *Ponto3D;
6
7 Ponto2D criar_Ponto2D (double x, double y);
8 Ponto3D criar_Ponto3D (double x, double y, double z);
9 void apagar_Ponto2D (Ponto2D *P);
10 void apagar_Ponto3D (Ponto3D *P);
11
12 double Ponto2D_getX (Ponto2D P);
13 double Ponto2D_getY (Ponto2D P);
14 void Ponto2D_setX (Ponto2D P, double x);
15 void Ponto2D_setY (Ponto2D P, double y);
16
17 double Ponto3D_getX (Ponto3D P);
18 double Ponto3D_getY (Ponto3D P);
19 double Ponto3D_getZ (Ponto3D P);
20 void Ponto3D_setX (Ponto3D P, double x);
21 void Ponto3D_setY (Ponto3D P, double y);
22 void Ponto3D_setZ (Ponto3D P, double z);
23
24 double euclidiana_2d (Ponto2D P, Ponto2D Q);
25 double euclidiana_3d (Ponto3D P, Ponto3D Q);
26
27#endif
```

A estrutura interna está oculta! O usuário não consegue saber COMO a struct st_Ponto2D e a struct st_Ponto3D são criadas internamente.

Isso esconde a estrutura de dados que implementa esses tipos abstratos! O usuário não consegue mais acessar as estruturas internas diretamente (ele não consegue nem mais saber quais são as estruturas internas).



Como a estrutura interna está oculta, a interface deve fornecer todos os comportamentos necessários.

Um TAD é definido apenas por seus comportamentos!



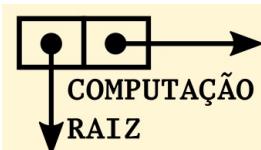
Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

```
1 #include "pontoTAD.h"
2 #include <math.h>
3 #include <stdlib.h>
4
5 struct st_Ponto2D
6 {
7     double coordX, coordY;
8 };
9
10 struct st_Ponto3D
11 {
12     double coordX, coordY, coordZ;
13 };
14
15 Ponto2D criar_Ponto2D (double x, double y)
16 {
17     Ponto2D T = malloc(sizeof(struct st_Ponto2D));
18     if (!T)
19         return NULL;
20     T->coordX = x;
21     T->coordY = y;
22     return T;
23 }
24 }
```



Na implementação da interface a estrutura de dados que implementa o TAD é definida, mas o usuário não consegue ver nem acessar.

A interface deve fornecer todos os comportamentos (operações) para que o cliente consiga utilizar o TAD.



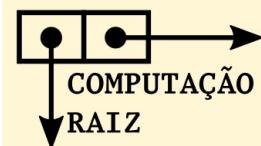
Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

```
25 Ponto3D criar_Ponto3D (double x, double y, double z)
26 {
27     Ponto3D T = malloc(sizeof(struct st_Ponto3D));
28     if (!T)
29         returne NULL;
30     T->coordX = x;
31     T->coordY = y;
32     T->coordZ = z;
33     return T;
34 }
35
36 void apagar_Ponto2D (Ponto2D *P)
37 {
38     if (*P)
39     {
40         free(*P);
41         *P = NULL;
42     }
43 }
44
45 void apagar_Ponto3D (Ponto3D *P)
46 {
47     if (*P)
48     {
49         free(*P);
50         *P = NULL;
51     }
52 }
```



pontoTAD.c

A interface deve fornecer todos os comportamentos (operações) para que o cliente consiga utilizar o TAD.

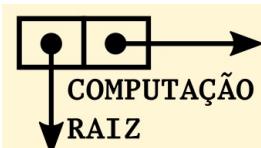


Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

```
54 double Ponto2D_getX (Ponto2D P)
55 {
56     return P->coordX;
57 }
58
59 double Ponto2D_getY (Ponto2D P)
60 {
61     return P->coordY;
62 }
63
64 void Ponto2D_setX (Ponto2D P, double x)
65 {
66     P->coordX = x;
67 }
68
69 void Ponto2D_setY (Ponto2D P, double y)
70 {
71     P->coordY = y;
72 }
73
74 double Ponto3D_getX (Ponto3D P)
75 {
76     return P->coordX;
77 }
78
79 double Ponto3D_getY (Ponto3D P)
80 {
81     return P->coordY;
82 }
```



A interface deve fornecer todos os comportamentos (operações) para que o cliente consiga utilizar o TAD.



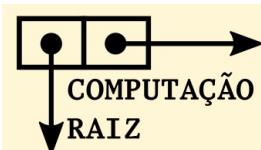
Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

```
84 double Ponto3D_getZ (Ponto3D P)
85 {
86     return P->coordZ;
87 }
88
89 void Ponto3D_setX (Ponto3D P, double x)
90 {
91     P->coordX = x;
92 }
93
94 void Ponto3D_setY (Ponto3D P, double y)
95 {
96     P->coordY = y;
97 }
98
99 void Ponto3D_setZ (Ponto3D P, double z)
100 {
101     P->coordZ = z;
102 }
```



pontoTAD.c

A interface deve fornecer todos os comportamentos (operações) para que o cliente consiga utilizar o TAD.



Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

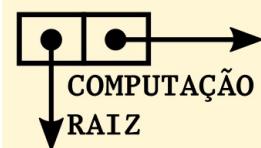
```
104 double euclidiana_2d (Ponto2D P, Ponto2D Q)
105 {
106     return sqrt(pow((P->coordX - Q->coordX), 2.0) +
107                 pow((P->coordY - Q->coordY), 2.0));
108 }
109
110 double euclidiana_3d (Ponto3D P, Ponto3D Q)
111 {
112     return sqrt(pow((P->coordX - Q->coordX), 2.0) +
113                 pow((P->coordY - Q->coordY), 2.0) +
114                 pow((P->coordZ - Q->coordZ), 2.0));
115 }
```



pontoTAD.c

A interface deve fornecer todos os comportamentos (operações) para que o cliente consiga utilizar o TAD.

```
gcc -std=c17 -Wall -Wpedantic -Wconversion -Werror -Wunused-result -c -o pontoTAD.o pontoTAD.c
```



Tipos de dados criados pelo programador: Tipo Abstrato de Dados (TAD)

```
1 #include "pontoTAD.h"
2 #include <stdio.h>
3
4 int main (void)
5 {
6     Ponto2D A = criar_Ponto2D(0, 0);
7     Ponto2D B = criar_Ponto2D(3, 4);
8
9     Ponto3D C = criar_Ponto3D(0, 0, 0);
10    Ponto3D D = criar_Ponto3D(3, 3, 3);
11
12    double dist2 = euclidiana_2d(A, B);
13    double dist3 = euclidiana_3d(C, D);
14
15    printf("Dist. entre A e B: %.2f\n", dist2);
16    printf("Dist. entre C e D: %.2f\n", dist3);
17
18    apagar_Ponto2D(&A);
19    apagar_Ponto2D(&B);
20    apagar_Ponto3D(&C);
21    apagar_Ponto3D(&D);
22 }
```

gcc -std=c17 -Wall -Wpedantic -Wconversion -Werror -Wunused-result -o cliente3 cliente3.c pontoTAD.o -lm

```
./cliente3
Dist. entre A e B: 3.00
Dist. entre C e D: 5.20
```



O cliente usa a interface para tudo, já que a estrutura interna do tipo abstrato de dado é oculta.

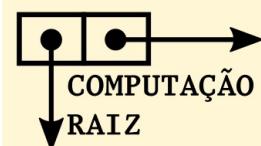
Tudo que o cliente precisa fazer, ele faz através dos comportamentos (operações) do tipo de dado!



Tipos Concretos x Tipos Abstrados

Por que os TADs são melhores do que os TCDs?

- TADs são definidos apenas pelo seu comportamento (criar, apagar, cálculos, etc.) e não pela estrutura interna.
- Como a estrutura interna está oculta, o cliente não consegue acessar diretamente nada da implementação e é obrigado a fazer tudo pelas operações da interface.
- Como a estrutura interna está oculta, o programador pode alterar completamente a estrutura interna sem causar nenhuma alteração nos programas clientes.
- Maior segurança através da barreira da abstração.



Não confunda TAD com Estruturas de Dados!

Uma dificuldade comum dos estudantes é confundir os conceitos de TAD e de Estruturas de Dados (pois realmente são muito relacionados). Mas atenção: **NÃO CONFUNDA TAD COM ESTRUTURA DE DADOS!**

- Um **TAD** é um tipo abstrato de dado criado pelo programador cuja principal característica é **ser definido por seu comportamento**, não por sua estrutura interna que fica totalmente oculta.
- Uma **Estrutura de Dados** é uma **estrutura de memória que armazena os dados** (valores) que estão manipulados por um algoritmo, de forma **organizada e otimizada**, para aumentar a eficiência do processamento que está sendo realizado.

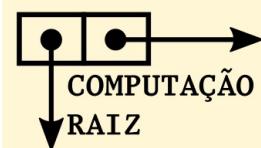


Não confunda TAD com Estruturas de Dados!

Uma dificuldade comum dos estudantes é confundir os conceitos de TAD e de Estruturas de Dados (pois realmente são muito relacionados). Mas atenção: **NÃO CONFUNDA TAD COM ESTRUTURA DE DADOS!**

Uma das fontes de confusão para os alunos é que nós:

- Podemos implementar um determinado TAD com diversas Estruturas de Dados, por exemplo, uma pilha (um TAD importante na computação) pode ser implementada, por exemplo, com as seguintes estruturas de dados:
 - Arrays estáticos
 - Arrays dinâmicos
 - Listas encadeadas
- Podemos criar TADs apenas para implementar uma determinada estrutura de dados, por exemplo: uma árvore binária de busca pode ser um TAD implementado sobre a estrutura de dados "árvore binária".

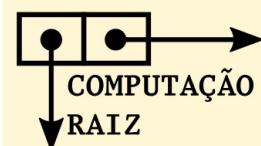


Não confunda TAD com Estruturas de Dados!

Uma dificuldade comum dos estudantes é confundir os conceitos de TAD e de Estruturas de Dados (pois realmente são muito relacionados). Mas atenção: **NÃO CONFUNDA TAD COM ESTRUTURA DE DADOS!**

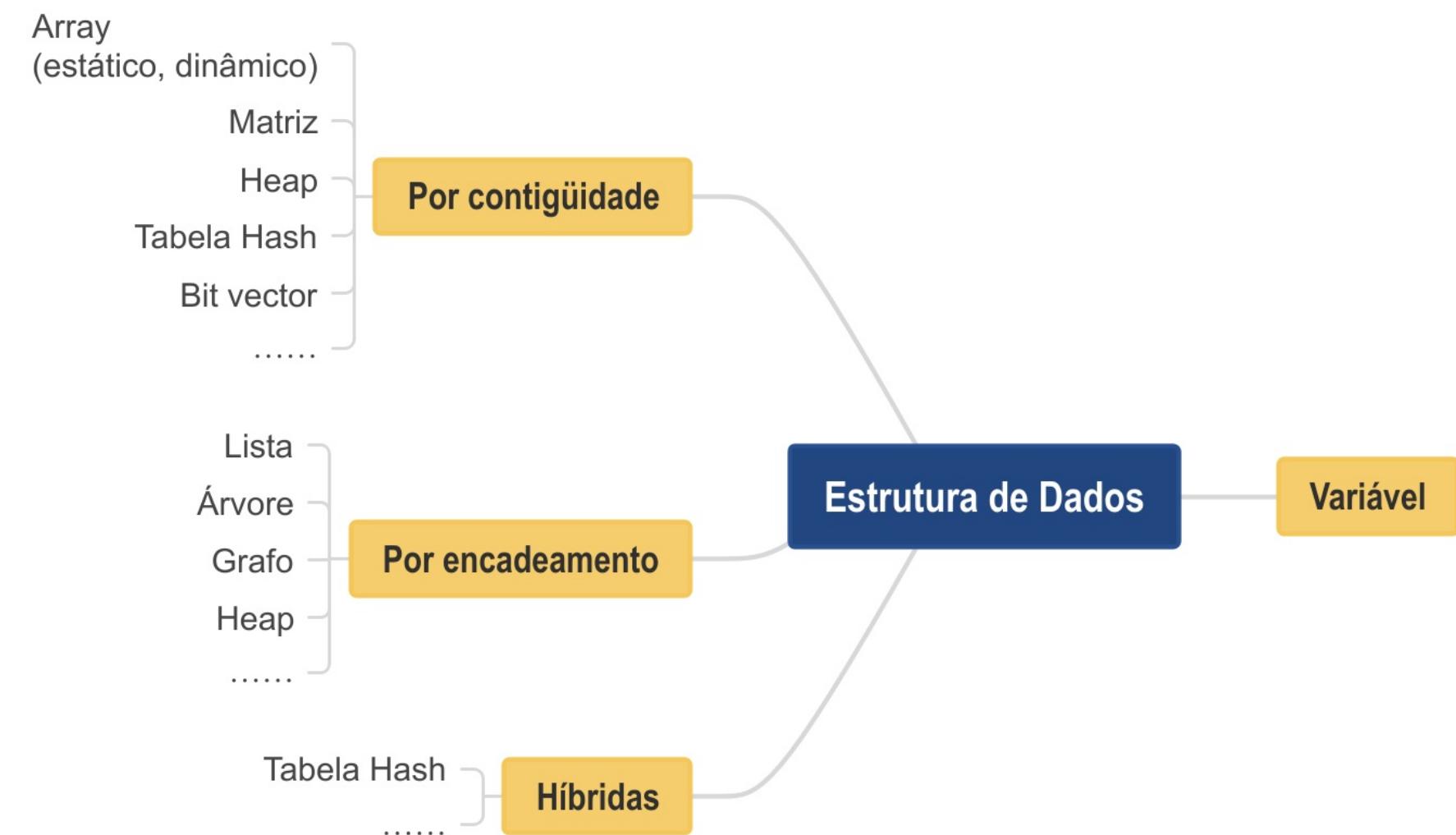
Se estiver em dúvida se um objeto é um TAD ou uma Estrutura de Dados, lembre-se do seguinte:

- a) Se **o objeto em questão está definido apenas em termos de seus comportamentos (operações), e a representação interna está oculta**, estamos diante de um **TAD**.
- b) Se, por outro lado, **o objeto mostra sua estrutura interna e podemos perceber e entender como os dados estão organizados diretamente na memória**, então estamos diante de uma **Estrutura de Dados** (ou um **TCD**).

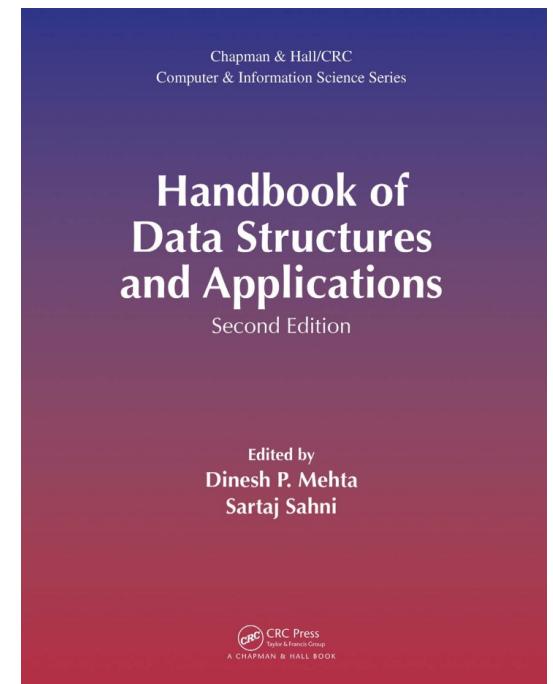


Estruturas de Dados x Tipos Abstratos de Dados

OK, já vimos algumas das mais importantes Estruturas de Dados da computação para organização linear e não-linear dos dados, de forma contígüa, por encadeamento ou formas híbridas:

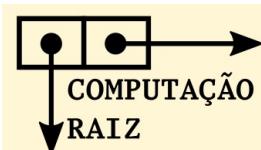


Não se engane... há muito mais estruturas de dados do que as exibidas aqui!



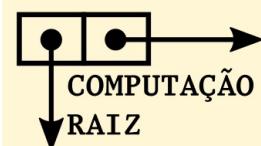
Mais de 1000 páginas apenas de estruturas de dados e TADs para as mais diversa aplicações.

E os TADs da computação? Vamos fazer uma pausa nas Estruturas de Dados para conhecer os TADs. Depois faremos um estudo geral.



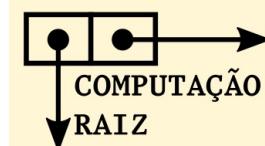
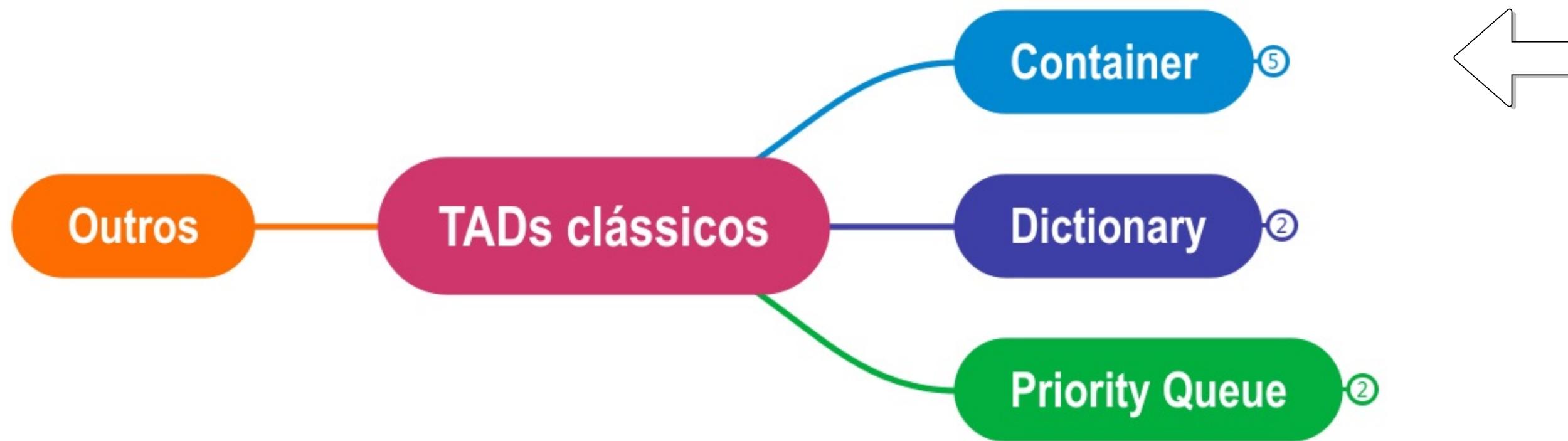
Principais TADs da computação

Da mesma maneira que as estruturas de dados, há inúmeros **TADs clássicos** na computação. Eles podem ser divididos, didaticamente, do seguinte modo:



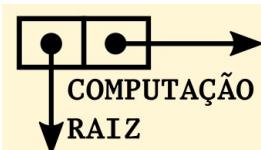
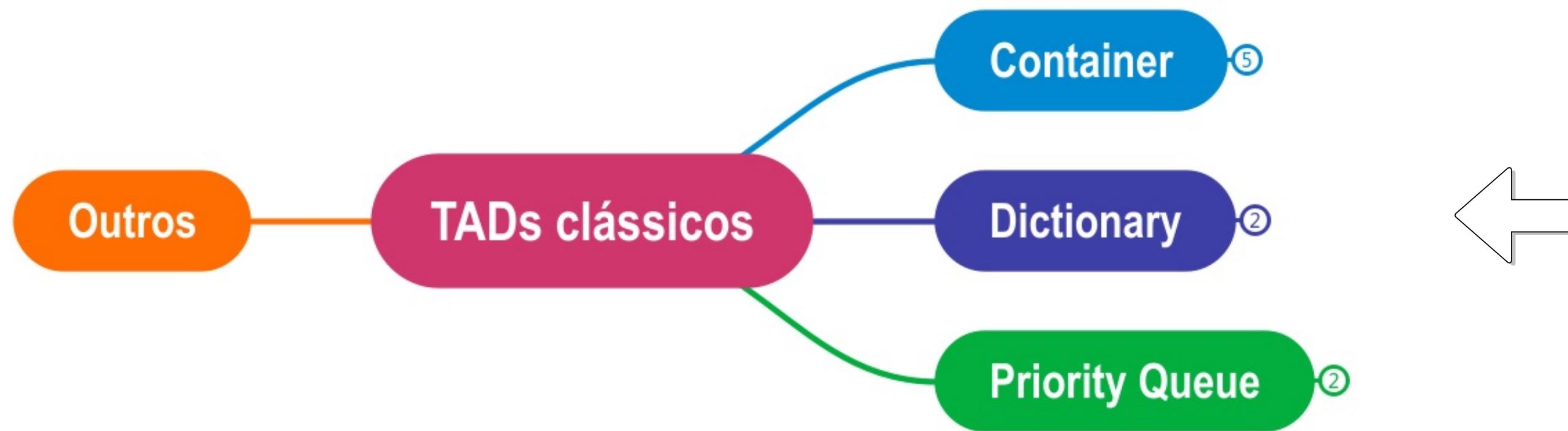
Principais TADs da computação

Container: consiste de um grupo de TADs que permitem o **armazenamento, inserção, remoção e busca de forma independente do conteúdo que está sendo armazenado**. Essas operações são baseadas na posição dos elementos, não no conteúdo do elemento.



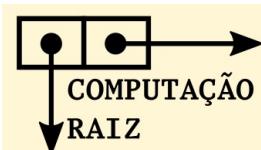
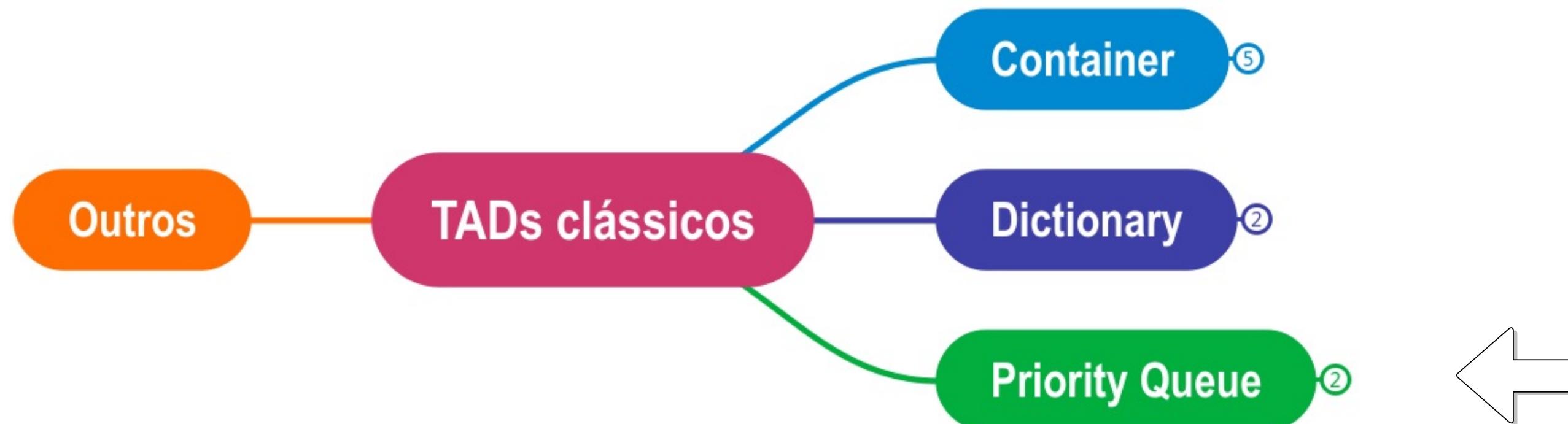
Principais TADs da computação

Dictionary: consiste de um grupo de TADs que permitem o **armazenamento, inserção, remoção e busca de acordo com o conteúdo que está armazenado**. Essas operações são baseadas em uma chave armazenada, que está associada a um valor.



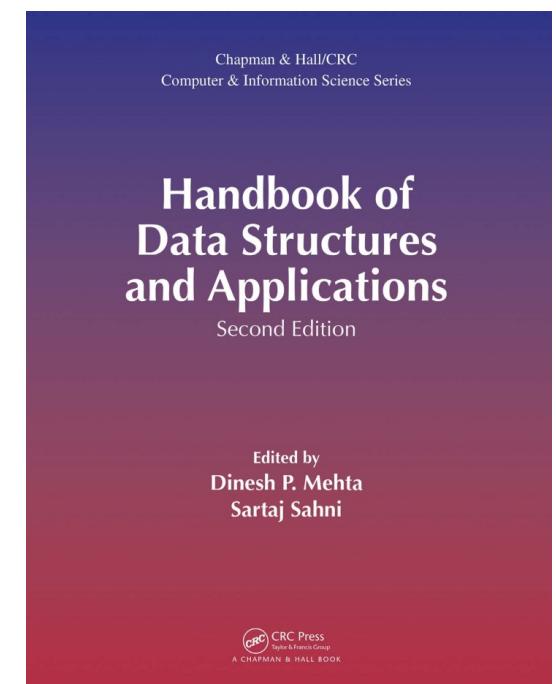
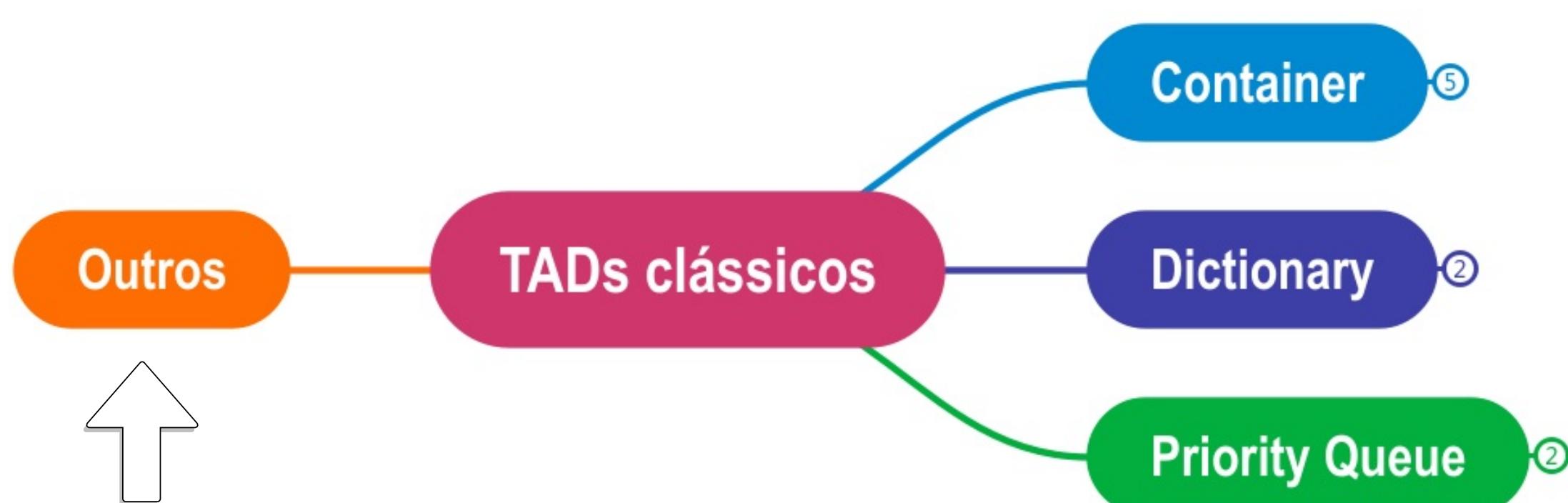
Principais TADs da computação

Priority Queue: consiste de um grupo de TADs que permitem o **armazenamento, inserção, remoção e busca de acordo com uma ordem pré-definida de prioridade**. Essas operações são baseadas no prioridade de um objeto armazenado.

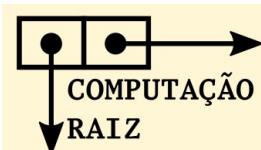


Principais TADs da computação

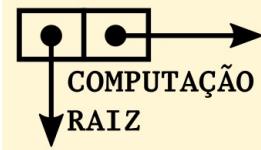
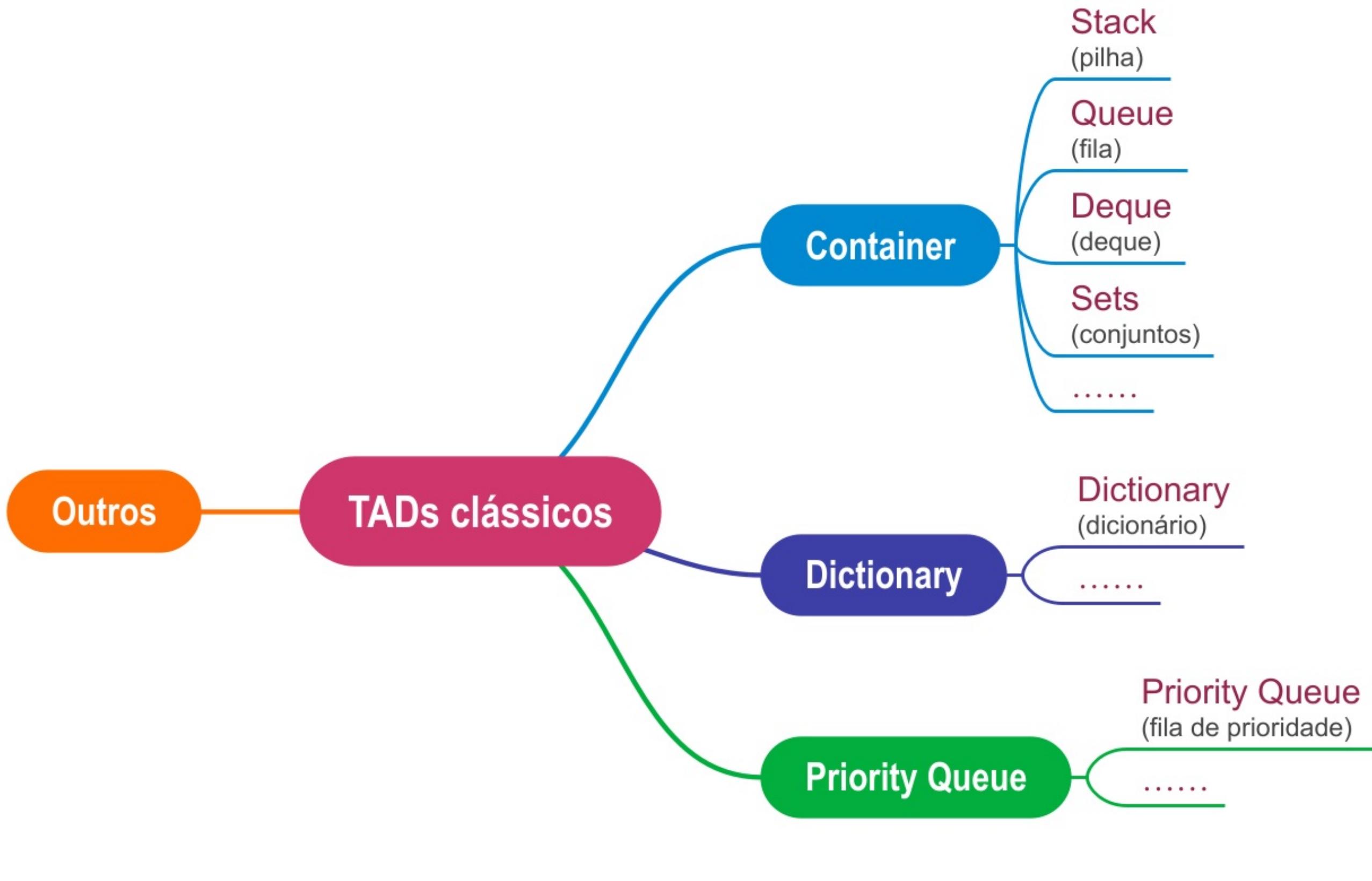
Outros: existem uma infinidade de TADs (e estruturas de dados) com uso especializado para determinadas aplicações (geometria plana e espacial, funcionais, concorrência...).



Mais de 1000 páginas apenas de estruturas de dados e TADs para as mais diversas aplicações.



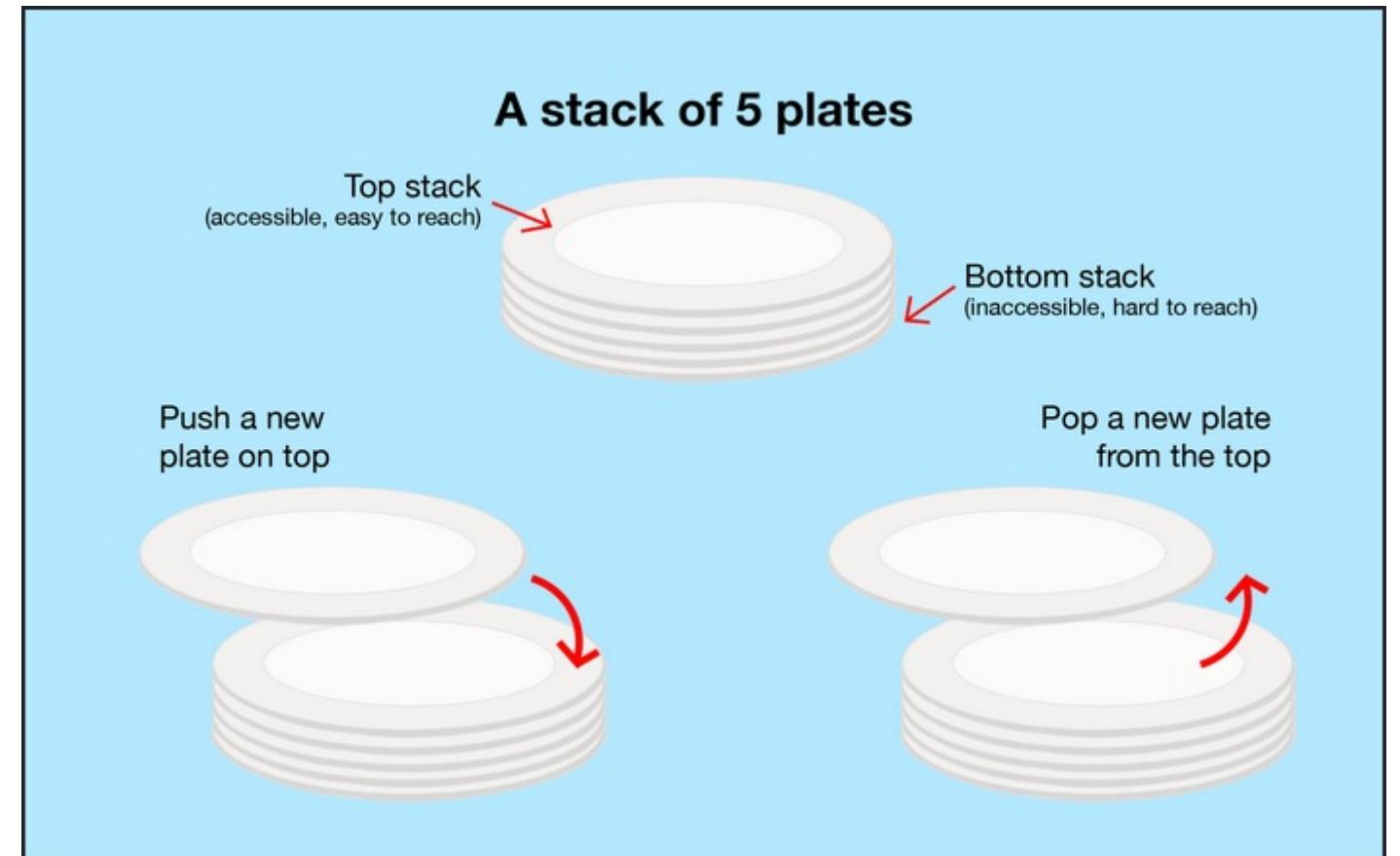
Principais TADs da computação



TAD Stack (pilha)

É um TAD da categoria dos **containers**, ou seja, é um TAD que permite **armazenar e recuperar dados independentemente de seu valor/conteúdo**.

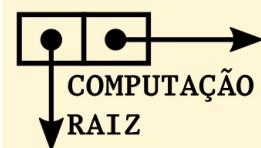
A principal característica do comportamento de uma pilha é que **os dados só podem ser retirados da pilha na ordem inversa em que foram adicionados**, ou seja, de modo **LIFO** (last in, first out).



Fonte: <https://visualgo.net/>

Os principais **comportamentos** do Stack são:

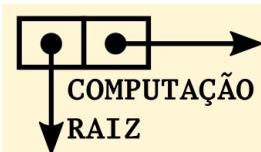
- push	insere um elemento no topo da pilha	O(1)
- pop	retorna e remove o elemento do topo da pilha	O(1)
- full/empty	verifica se a pilha está cheia ou vazia	O(1)
- top/peek	retorna sem remover o elemento do topo	O(1)
- size	número de elementos na pilha	O(1)



TAD Stack (pilha)

Principais usos do Stack:

- modelar comportamento LIFO: ordem inversa de chegada, item mais recente...
- remover estado encapsulado
- busca em profundidade iterativa em grandes grafos (em listas de adjacência)
- matemática com parênteses balanceados; validação de expressões
- conversão de notação infix → postfix (shunting yard, de Dijkstra)
- calculadoras pós-fixadas
- "stack monotônico": próximo/anterior maior/menor; maior área em histogramas; maior/menor em todos os subarrays de comprimento fixo; ...
- backtracking com reconstrução de caminhos, geração de permutações, ...
- simulação de chamadas recursivas
- algoritmos para grafos específicos (Kosaraju, Tarjan)
- algoritmos geométricos: convex hull (Graham Scan)
- busca de componentes fortemente conectados em grafos
- validação de tags html/xml
- outros: undo/redo; inverter strings, listas arrays;



TAD Stack (pilha)

Implementação do Stack:

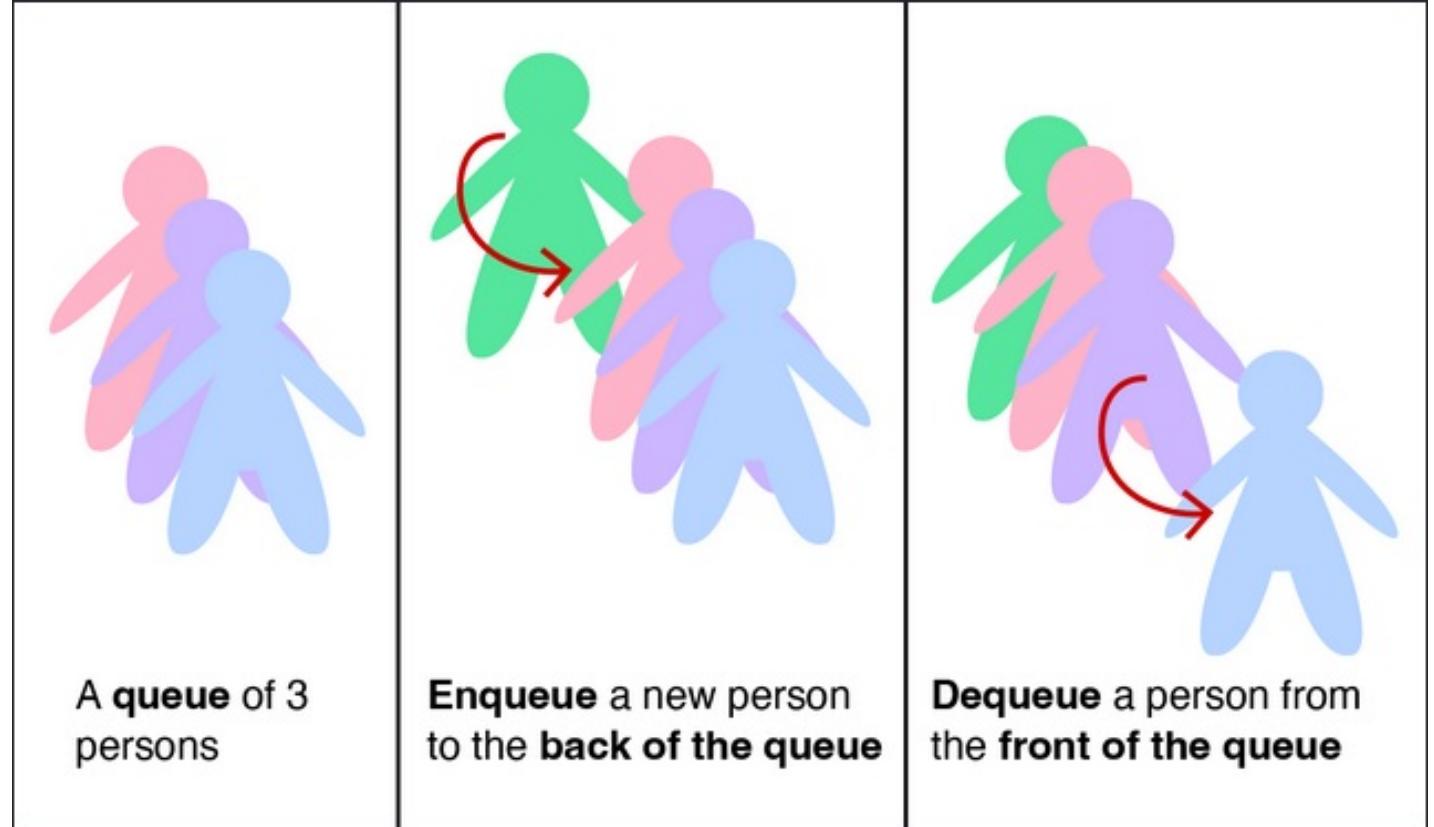
- array estático
- array dinâmico
- lista encadeada



TAD Queue (fila)

É um TAD da categoria dos **containers**, ou seja, é um TAD que permite **armazenar e recuperar dados independentemente de seu valor/conteúdo**.

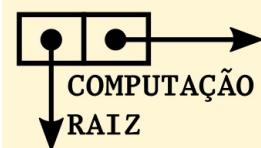
A principal característica do comportamento de uma fila é que **os dados só podem entrar no final da fila, e só podem sair no começo da fila**, ou seja, de modo **FIFO** (first in, first out).



Fonte: <https://visualgo.net/>

Os principais **comportamentos** do Queue são:

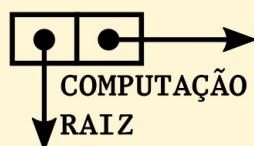
- enqueue	insere um elemento no fim da fila	$O(1)$
- dequeue	retorna e remove o elemento do início da fila	$O(1)$ até $O(n)$
- full/empty	verifica se a fila está cheia ou vazia	$O(1)$
- front/peek	retorna, sem remover, o elemento do início	$O(1)$
- size	quantidade de elementos na fila	$O(1)$



TAD Queue (fila)

Principais usos do Queue:

- modelar montes de cartas de baralho
- simulação de processos e modelagem FIFO: filas, caixas, round-robin, montes de cartas de baralho, trânsito de veículos, ...
- janelas deslizantes: soma/contagens
- busca "breadth-first" em grafos e grades: menor caminho em grafo não ponderado, componentes conectados, teste de bipartição, travessia nível a nível, menor caminho em grafo de estados
- ordenação topológica em grafo direcionado acíclico: dependência de tarefas, seqüências de montagem/construção
- percorrer árvores nível por nível
- flood fill e preenchimento de regiões
- outros



TAD Queue (fila)

Implementação do Queue:

- array estático
 - inserção: $O(1)$; remoção: $O(n)$
- array estático circular (ring buffer)
 - inserção: $O(1)$; remoção: $O(1)$
- array dinâmico: $O(1)$ amortizado
 - a cópia do array é $O(n)$
- lista encadeada: $O(1)$



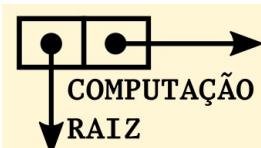
TAD Deque (deque: "double-ended queue")

É um TAD da categoria dos **containers**, ou seja, é um TAD que permite **armazenar e recuperar dados independentemente de seu valor/conteúdo**.

A principal característica do comportamento de um deque é que **os dados podem ser inseridos e removidos em ambas as pontas do deque**, ou seja, permite inserção/remoção no início e no fim.

Os principais **comportamentos** do Deque são:

- push back/front	insere um elemento no fim/início	O(1)
- pop back/front	retorna e remove o elemento fim/início	O(1)
- front/back	retorna sem remover o elemento	O(1)
- full/empty	cheio ou vazio	O(1)
- size	número de elementos	O(1)



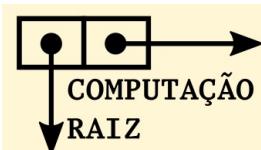
TAD Deque (deque: "double-ended queue")

Principais usos do Deque:

- modelar situações onde as duas pontas precisam ser usadas
- busca "breadth-first" em grafos 0/1 ponderados
- algoritmos de janela deslizante: máximo/mínimo
- monotonic queue
- construção/verificação de palíndromos
- simular arrays/buffers circulares
- outros

Implementação do Deque:

- array estático
 - inserção: $O(1)$; remoção: $O(1)$
- array dinâmico: $O(1)$ (a cópia do array é $O(n)$)
- lista duplamente encadeada: $O(1)$



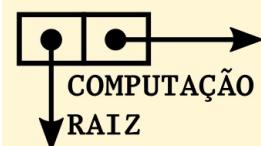
TAD Set (conjunto)

É um TAD da categoria dos **containers** (considerados em uma visão mais ampla), ou seja, é um TAD que permite **armazenar coleções de elementos únicos**. Alguns autores classificam os Sets na categoria dos dicionários, e outros dizem que Sets foram uma categoria separada das demais.

Os conjuntos (mais estritamente falando, os subconjuntos) são **coleções não ordenadas de elementos únicos retirados de um dado conjunto universo U**.

Os principais **comportamentos** do Set são:

- | | |
|-----------------------------|---|
| - member(x, S) | x pertence a S? |
| - union(A, B) | obtém a união dos conjuntos A e B |
| - intersection(A, B) | obtém a interseção dos conjuntos A e B |
| - insert(x, S) | insere o elemento x no conjunto S |
| - remove(x, S) | remove o elemento x do conjunto S |



TAD Set (conjunto)

- Principais usos do Set:

- modelar situações onde subconjuntos são necessários
- garantir unicidade de elementos em uma coleção
- outros

Implementação do Set:

- bit vectors
- bit masks
- dictionary (hash table)
- outras



TAD Dictionary (dicionário)

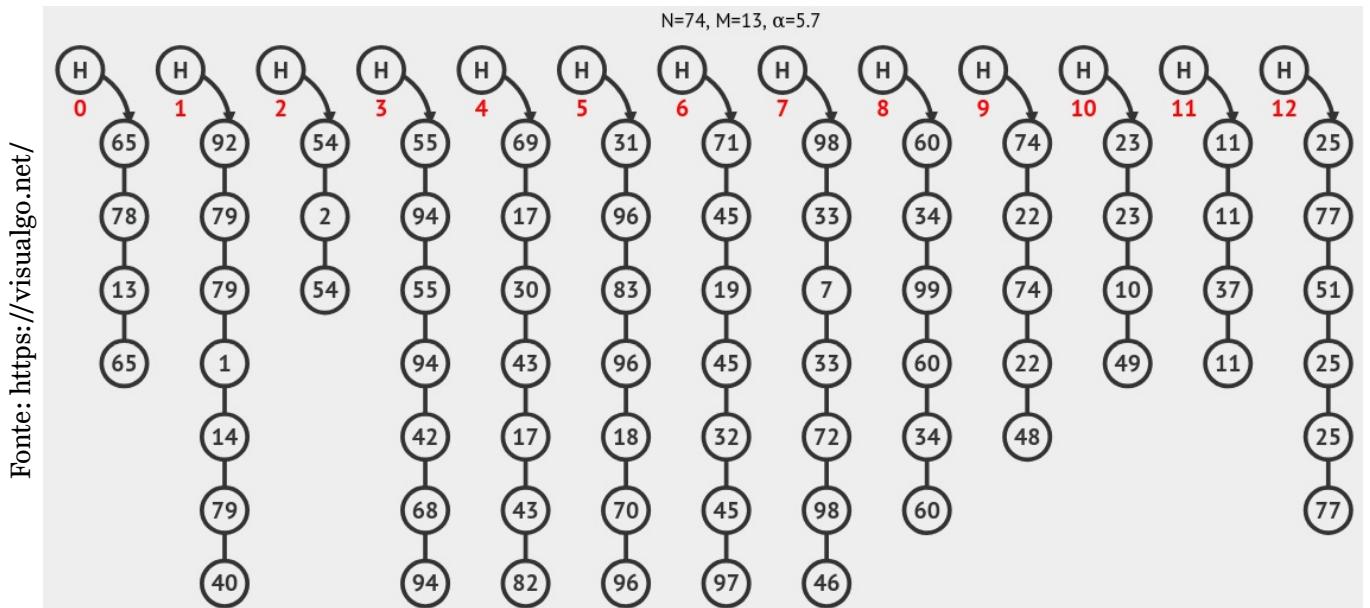
É um TAD da categoria dos **dictionaries**, e permitem o **armazenamento, inserção, remoção e busca de acordo com o conteúdo que está armazenado**. Essas operações são baseadas no valor de uma chave armazenada, sendo que cada chave está associada a um valor.

A principal característica do comportamento de um dicionário é que **os dados são armazenados em PARES <chave:valor>**, e **os dados só podem ser acessados pela chave associada**. Não há valor diferente com chave igual.

Os principais **comportamentos** do Dictionary são:

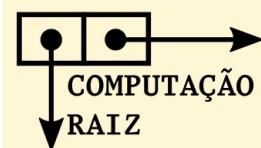
- insert
- delete
- search

insere um par chave:valor no dicionário
remove um par chave:valor do dicionário
retorna o valor associado a uma chave, se a chave existir



Também são conhecidos por:

- arrays associativos
- table
- map
- symbol table
- ...
- container associativo
- associative table
- key-value store



TAD Dictionary (dicionário)

- Principais usos do Dictionary:

- modelar situações onde valores estão associados à chaves
- outros

Implementação do Dictionary:

- **Static Dictionary**: criados uma vez e não se alteram mais
 - arrays ordenados/não ordenados
- **Semi-Dynamic Dictionary**: inserção e busca (sem remoção)
 - listas encadeadas ordenadas/não ordenadas
 - hash table (com endereçamento aberto ou com encadeamento)
- **Fully Dynamic Dictionary**: inserção, busca, remoção
 - hash table (com encadeamento)
 - árvores diversas (splay, AVL, RB, ...)



TAD Priority Queue (fila de prioridade)

É um TAD da categoria dos **priority queues**, ou seja, é um TAD que permite o **armazenamento, inserção, remoção e busca de acordo com uma ordem pré-definida de prioridade**. Essas operações são baseadas no prioridade de um objeto armazenado

A principal característica do comportamento de uma fila de prioridade é que **os dados são armazenados junto com uma prioridade, e a remoção se dá sempre pelo dado de maior prioridade**. Pode-se ter elementos com igual prioridade.

Os principais **comportamentos** do Priority Queue são:

- insert insere um item com uma determinada prioridade
- maximum retorna o item com a maior prioridade
- extract retorna e remove o item com a maior prioridade



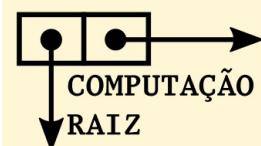
TAD Priority Queue (fila de prioridade)

- Principais usos do Priority Queue:

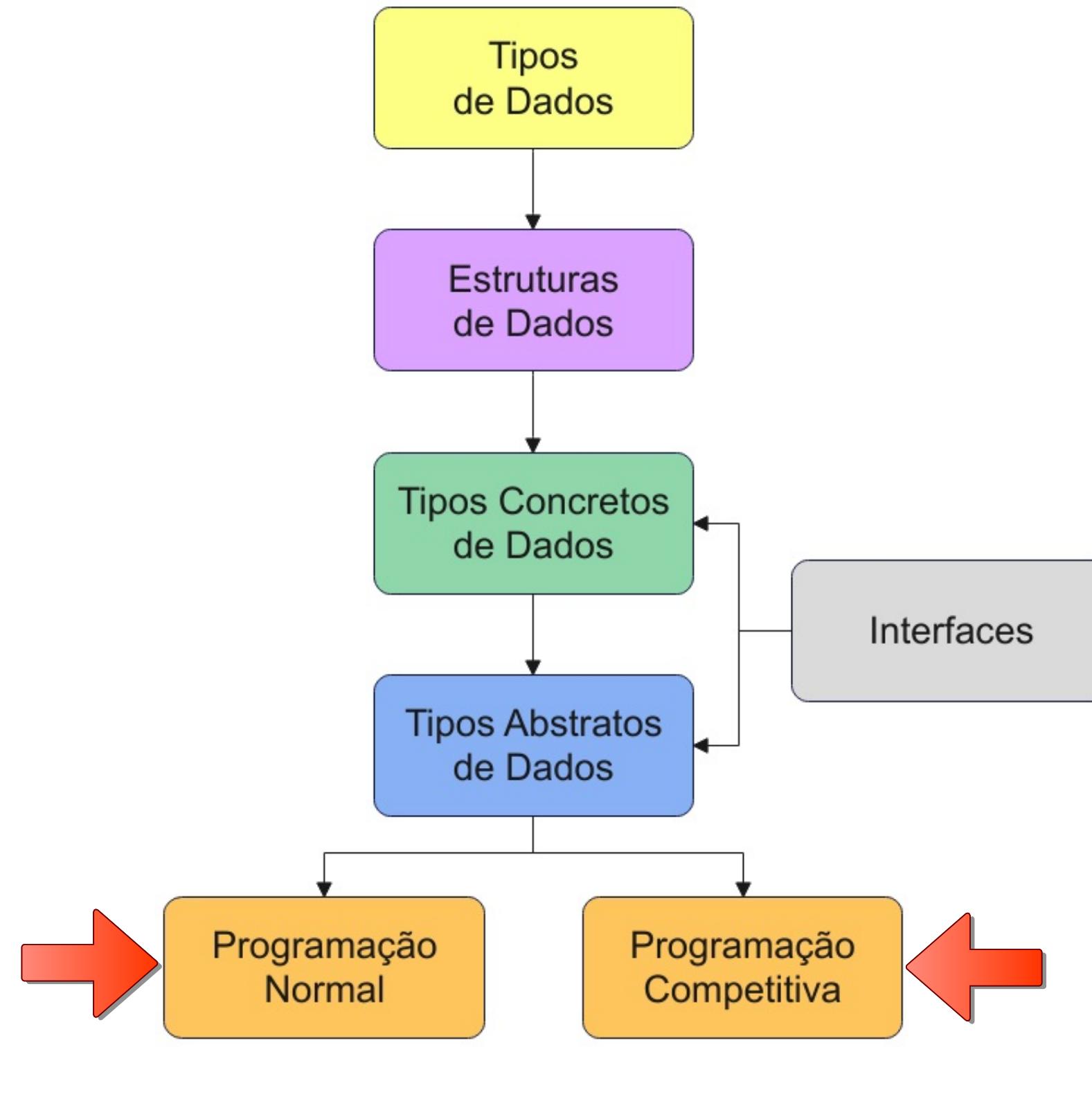
- algoritmos gulosos (agendamento de tarefas, k-ésimo melhor/maior/menor, ...)
- algoritmos em grafos (Dijkstra, Prim)
- ordenação parcial, mesclagem de listas
- manutenção de conjuntos dinâmicos
- compressão de Huffman
- caminhos k-curtos / k-vizinhos mais próximos
- sliding window

Implementação do Priority Queue:

- heap
- árvore binária balanceada



Estruturas de Dados e TADs na programação normal e competitiva



Programação "normal"

O objetivo aqui é criar um **programa eficiente que será mantido ao longo do tempo**, com um bom projeto de engenharia de software. Assim:

- Usamos interfaces
- Evitamos TCDs
- Criamos TADs específicos e genéricos
- Modularizamos código
- Documentamos tudo
- Criamos nossas próprias bibliotecas
- Criamos nossas próprias estruturas de dados
- Fazemos as coisas devagar, com paciência e critério
- etc...



Fonte: Getty Images, no Unsplash

O objetivo de disciplinas como **Estruturas de Dados I e II** é ensinar como criar essas coisas "na unha", para que você entenda todos os detalhes de implementação e saiba como criar do zero essas estruturas e TADs.



Programação "competitiva"

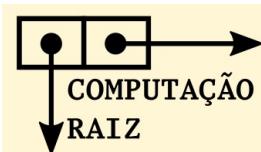
O objetivo aqui é criar um **programa de uso único**, que não será utilizado por mais ninguém (inclusive você) depois da competição, na maior velocidade possível para obter uma resposta correta com o menor número de penalidades. Assim:

- Não criamos interfaces
- Não criamos TADs (no máximo TCDs)
- Não modularizamos código, não documentamos
- Não criamos nossas próprias bibliotecas
- Criamos nossas próprias estruturas de dados apenas se a biblioteca da linguagem escolhida não suportar (ex.: grafos em C/C++)
- etc...



Fonte: gerada pelo Gemini

O objetivo de **Treinamentos para Maratonas**, como este, é ensinar **como escolher a melhor estrutura de dados e TADs**, já disponível na **biblioteca da linguagem C++**, e **usar essas ferramentas com eficiência na resolução de um problema computacional**. Espera-se que você já conheça essas estruturas.



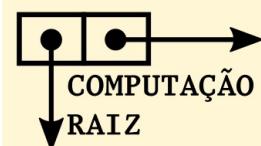
E por que essa teoria toda?

Porque quem não entende, no mínimo, a diferença entre **tipos de dados**, **estruturas de dados**, **tipos concretos de dados** e **tipos abstratos de dados** tem maior dificuldade da computação mais avançada e, em especial, na maratona de programação.

Porque, ao usar as bibliotecas de C++ você estará usando TADs de alto nível, ou seja, você **não precisará implementar as estruturas subjacentes** mas, no mínimo, você deve **saber como esses TADs são implementados e a complexidade de cada um** (Big-O) para escolher a melhor ferramenta para resolver um problema.

Porque, apesar de C++ ter muitos TADs prontos, até o momento **não existem TADs para todas as situações** (grafos, por exemplo) e, nessa situação, temos que criar nossas próprias estruturas e TCDs.

Porque você precisa dessa base para **estudar por conta própria** e conseguir **resolver os problemas de programação!**



Próxima aula:

Como usar essas estruturas de dados e TADs em C++!

Até lá!

