

Estrutura de Dados II (ED2)

Aula 01 – Regras do jogo e Introdução

Departamento de Informática (DI)
Centro Tecnológico (CT)
Universidade Federal do Espírito Santo (UFES)

(Material baseado nos slides do Professor Eduardo Zambon)

Contato

- **Nome:** Giovanni Comarela
- **E-mail:** `gc@inf.ufes.br`
- **Telefone:** (27) 4009-XXXX
- **Escritório:** CT-7 Sala 28

- Ambiente participativo e interativo

Sobre as aulas

- Ambiente participativo e interativo
- Aulas são planejadas assumindo que haverá perguntas

Sobre as aulas

- Ambiente participativo e interativo
- Aulas são planejadas assumindo que haverá perguntas
- Não há perguntas idiotas.

Sobre as aulas

- Ambiente participativo e interativo
- Aulas são planejadas assumindo que haverá perguntas
- Não há perguntas idiotas. **False!**

Sobre as aulas

- Ambiente participativo e interativo
- Aulas são planejadas assumindo que haverá perguntas
- Não há perguntas idiotas. **False!**
- Há perguntas idiotas, mas elas são bem-vindas!

Sobre as aulas

- Ambiente participativo e interativo
- Aulas são planejadas assumindo que haverá perguntas
- Não há perguntas idiotas. **False!**
- Há perguntas idiotas, mas elas são bem-vindas!
- Não há tolerância para:
 - Destratar, diminuir ou depreciar uma pessoa tirando dúvida
 - Qualquer tipo de discriminação

Sobre as aulas

- Ambiente participativo e interativo
- Aulas são planejadas assumindo que haverá perguntas
- Não há perguntas idiotas. **False!**
- Há perguntas idiotas, mas elas são bem-vindas!
- Não há tolerância para:
 - Destratar, diminuir ou depreciar uma pessoa tirando dúvida
 - Qualquer tipo de discriminação
- Sobre política...

Sobre as aulas

- Ambiente participativo e interativo
- Aulas são planejadas assumindo que haverá perguntas
- Não há perguntas idiotas. **False!**
- Há perguntas idiotas, mas elas são bem-vindas!
- Não há tolerância para:
 - Destratar, diminuir ou depreciar uma pessoa tirando dúvida
 - Qualquer tipo de discriminação
- Sobre política...
- Eu só preciso de um ambiente civilizado (silêncio e ordem):
 - Celular em aula?
 - Dormir em aula?

Sobre as dúvidas

- ~~Em sala de aula~~ durante aulas síncronas
- AVA! Discussões são encorajadas; apenas não postem as soluções
- Dúvidas não serão elucidadas por e-mail

É obrigatória! A presença será aferida em aulas síncronas e assíncronas.

- Síncronas: Formulário do Google. Se não puder participar por problemas técnicos, haverá 48 para submeter um fichamento sobre o tema exposto (haverá uma explicação no AVA e material de leitura indicado para tal tarefa);
- Assíncrona: A presença será aferida por entregáveis (fichamento ou exercícios).

Sobre Colaboração Indevida

Não paguem para ver!

Preciso lembrar das disciplinas de programação anteriores?

Óbvio!

Ponteiros, Ponteiros para `void`, Ponteiros para funções, ...

Sobre as avaliações (vide Plano de Ensino)

Trabalhos Práticos

- T1 – 10
- T2 – 10
- T3 – 10

$$MP = \frac{T1 + T2 + T3}{3}$$

Referência (vide plano de ensino)

Referência principal

Algorithms in C

Robert Sedgewick

3rd edition

Atenção

É impossível cobrir todo conteúdo do livro em sala. Toda aula terá uma leitura associada, a qual é obrigatória e de responsabilidade dos alunos.

Cronograma

No AVA, assim como todo o conteúdo do curso.

O cronograma poderá ser alterado durante o semestre.
Alterações serão mencionadas em sala de aula.

Posso gravar a aula?

- Slides e roteiros serão disponibilizados no AVA
- A gravação para uso pessoal é permitida
- Vocês não têm minha autorização para distribuir ou publicar material gravado em aula.

- 1 Esse curso vai ser difícil?

- 1 Esse curso vai ser difícil? Se você fizer os exercícios e leituras recomendados nos tempos recomendados, não.

- 1 Esse curso vai ser difícil? Se você fizer os exercícios e leituras recomendados nos tempos recomendados, não.
- 2 Vou precisar ler o livro e estudar em casa para passar?

- 1 Esse curso vai ser difícil? Se você fizer os exercícios e leituras recomendados nos tempos recomendados, não.
- 2 Vou precisar ler o livro e estudar em casa para passar? Sim.

- **Estruturas de dados (EDs)** são componentes fundamentais de qualquer sistema de computação.
- **Aula de hoje:** exemplos de como diferentes EDs podem impactar no desempenho de programas.
- **Objetivos:** motivar o estudo aprofundado de diferentes EDs.

Referências

Chapter 1 – Introduction

R. Sedgewick

Curso de ED2 (AKA TBO - Técnicas de Busca e Ordenação):

- Curso de nível intermediário em EDs.
(Continuação direta de ED1.)
- Foco em programação com aplicação em solução de problemas.
- **Algoritmo**: método para resolver um problema.
- **Estrutura de dados**: método para armazenar informação.

Curso de ED2 (AKA TBO - Técnicas de Busca e Ordenação):

- Curso de nível intermediário em EDs.
(Continuação direta de ED1.)
- Foco em programação com aplicação em solução de problemas.
- **Algoritmo**: método para resolver um problema.
- **Estrutura de dados**: método para armazenar informação.

Tópicos e estruturas que serão abordados:

- **Tipos de dados**: pilha, fila (simples e de prioridades), *bags*, *union-find*.
- **Ordenação**: *quicksort*, *mergesort*, *heapsort*, *radix sorts*.
- **Busca**: BST (*binary search tree*), árvores rubro-negras, tabelas *hash*.

Por que estudar algoritmos?

O seu impacto é amplo e profundo:

- **Biologia:** Projeto do genoma humano, enovelamento de proteínas, ...
- **Física:** simulação de partículas, astro-física, ...
- **Computadores:** *Layout* de circuitos, sistemas de arquivos, compiladores, ...
- **Computação gráfica:** filmes, jogos, realidade virtual, ...
- **Multimídia:** MP3, JPEG, DivX, HDTV, reconhecimento de padrões, ...
- **Internet:** Busca na *Web*, roteamento de pacotes, compartilhamento de arquivos distribuídos, ...

Por que estudar algoritmos?

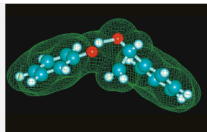
Para desvendar os mistérios do universo.

“Modelos computacionais que simulam sistemas reais são cruciais para a maioria das descobertas na química hoje... Hoje o computador é tão importante para um químico quanto um tubo de ensaio.”

— Vencedores do Prêmio Nobel de Química



Martin Karplus, Michael Levitt, and Arieh Warshel



Por que estudar algoritmos?

Para entender o mundo atual.

Algoritmo para manipular cotações já mexeu com 4% do mercado dos EUA

“Parece filme: um algoritmo de High-Frequency Trading invade a bolsa e faz milhares de investidores perderem suas economias. Algo similar aconteceu nos EUA semana passada, após um único robô postar e cancelar milhares de ordens em frações de segundos - em o que acredita-se ser uma tentativa de manipular o mercado. Isso foi o suficiente para se tornar 4% de todas as ordens nas bolsas dos EUA na semana passada, sem realizar um único negócio.”

Por que estudar algoritmos?

Para se tornar um programador proficiente.

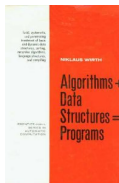
“De fato, eu acho que a diferença entre um mau programador e um bom programador é o que ele/ela considera mais importante: o seu código ou as suas estruturas de dados. Programadores ruins se preocupam com código. Programadores bons se preocupam com as estruturas de dados e as suas relações.”

— Linus Torvalds



“Algorithms + Data Structures = Programs.”

— Niklaus Wirth



Por que estudar algoritmos?



Passos para se desenvolver um algoritmo utilizável na prática:

- Modelar o problema.
- Encontrar um algoritmo para resolver.
- Rápido o suficiente? Consome muita memória?
- Se não, descobrir por quê.
- Encontrar uma forma de diminuir consumo de recursos.
- Iterar até satisfeito.

Passos para se desenvolver um algoritmo utilizável na prática:

- Modelar o problema.
- Encontrar um algoritmo para resolver.
- Rápido o suficiente? Consome muita memória?
- Se não, descobrir por quê.
- Encontrar uma forma de diminuir consumo de recursos.
- Iterar até satisfeito.

Pilares para estudo dos algoritmos:

- Método científico.
- Análise matemática.
- Análise empírica (experimental).

Problema da conectividade dinâmica

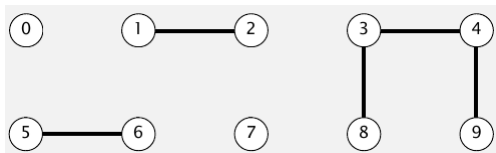
Dado um conjunto de N objetos, suportar duas operações:

- **Conectar** dois objetos.
- **Testar** se existe um caminho ligando dois objetos.

Problema da conectividade dinâmica

Dado um conjunto de N objetos, suportar duas operações:

- **Conectar** dois objetos.
- **Testar** se existe um caminho ligando dois objetos.



Conectar 4 e 3

Conectar 3 e 8

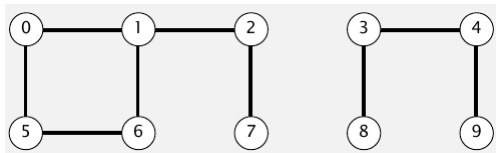
Conectar 6 e 5

Conectar 9 e 4

Conectar 2 e 1

Testar 0 e 7 ✗

Testar 8 e 9 ✓



Conectar 5 e 0

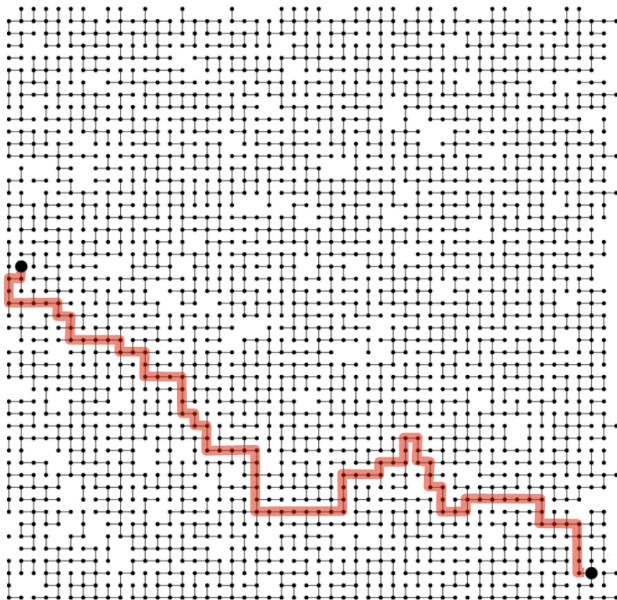
Conectar 7 e 2

Conectar 6 e 1

Um exemplo maior – Há um caminho?



Um exemplo maior – Há um caminho?



Modelando os objetos

Aplicações exigem a manipulação de variados tipos de objetos

- *Pixels* em uma foto digital.
- Computadores em uma rede.
- Amigos em uma rede social.
- Transistores em um chip.
- Elementos em um conjunto.
- Variáveis em um programa.

Modelando os objetos

Aplicações exigem a manipulação de variados tipos de objetos

- *Pixels* em uma foto digital.
- Computadores em uma rede.
- Amigos em uma rede social.
- Transistores em um chip.
- Elementos em um conjunto.
- Variáveis em um programa.

É conveniente numerar os objetos de 0 a $N - 1$:

- Permite usar o identificador do objeto como um índice em um *array*.
- Camada de abstração que permite ignorar detalhes desnecessários do problema.
- Não atrapalha pois é possível usar uma **tabela de símbolos** para se traduzir nomes para inteiros (aulas futuras).

Assumimos que “ser conectado com” é uma **relação de equivalência**:

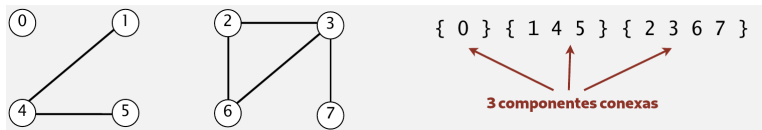
- **Reflexividade**: p é conectado com p .
- **Simetria**: se p é conectado com q então q é conectado com p .
- **Transitividade**: se p é conectado com q e q é conectado com r , então p é conectado com r .

Modelando as conexões

Assumimos que “ser conectado com” é uma **relação de equivalência**:

- **Reflexividade**: p é conectado com p .
- **Simetria**: se p é conectado com q então q é conectado com p .
- **Transitividade**: se p é conectado com q e q é conectado com r , então p é conectado com r .

Componente conexa: conjunto **maximal** de objetos que são mutuamente conectados.



Implementando as operações

Vamos implementar as seguintes operações:

Implementando as operações

Vamos implementar as seguintes operações:

- *Find*: Em qual componente o objeto p está?

Implementando as operações

Vamos implementar as seguintes operações:

- *Find*: Em qual componente o objeto p está?
- *Connected*: Os objetos p e q estão no mesmo componente?

Implementando as operações

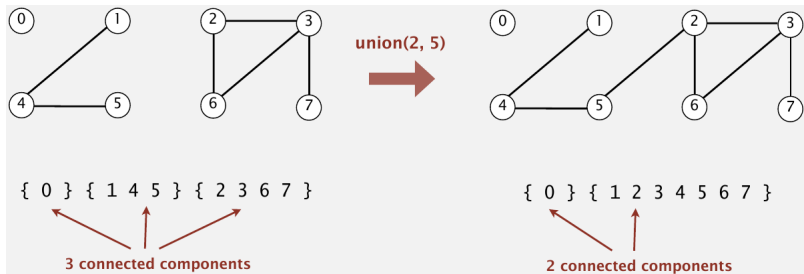
Vamos implementar as seguintes operações:

- *Find*: Em qual componente o objeto p está?
- *Connected*: Os objetos p e q estão no mesmo componente?
- *Union*: Substitua as componentes contendo os objetos p e q pela sua união.

Implementando as operações

Vamos implementar as seguintes operações:

- **Find:** Em qual componente o objeto p está?
- **Connected:** Os objetos p e q estão no mesmo componente?
- **Union:** Substitua as componentes contendo os objetos p e q pela sua união.



Tipo de dados (API) *union-find*

Objetivo: Projetar uma estrutura de dados eficiente.

- Número de objetos N pode ser muito grande.
- Número de operações M pode ser muito grande.
- Operações de *union* e *find* podem ser misturadas.

Tipo de dados (API) *union-find*

Objetivo: Projetar uma estrutura de dados eficiente.

- Número de objetos N pode ser muito grande.
- Número de operações M pode ser muito grande.
- Operações de *union* e *find* podem ser misturadas.

Arquivo `UF.h`:

```
#include <stdbool.h>

// Inicializa estrutura com N objetos numerados de 0 a N-1.
void UF_init(int N);
// Adiciona uma conexao entre p e q.
void UF_union(int p, int q);
// Retorna o identificador do componente de p (entre 0 a N-1).
int UF_find(int p);
// Os objetos p e q estao no mesmo componente?
bool UF_connected(int p, int q);
```

Tipo de dados (API) *union-find*

Objetivo: Projetar uma estrutura de dados eficiente.

- Número de objetos N pode ser muito grande.
- Número de operações M pode ser muito grande.
- Operações de *union* e *find* podem ser misturadas.

Arquivo `UF.h`:

```
#include <stdbool.h>

// Inicializa estrutura com N objetos numerados de 0 a N-1.
void UF_init(int N);
// Adiciona uma conexao entre p e q.
void UF_union(int p, int q);
// Retorna o identificador do componente de p (entre 0 a N-1).
int UF_find(int p);
// Os objetos p e q estao no mesmo componente?
bool UF_connected(int p, int q);
```

```
bool UF_connected(int p, int q) {
    return UF_find(p) == UF_find(q);
}
```


- Recebe o número N de objetos como argumento.
- Repete:
 - Lê um par de inteiros de `stdin`.
 - Se eles ainda não estão conectados, faz a união e imprime o par em `stdout`.

- Recebe o número N de objetos como argumento.
- Repete:
 - Lê um par de inteiros de `stdin`.
 - Se eles ainda não estão conectados, faz a união e imprime o par em `stdout`.

Arquivo `client.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include "UF.h"

int main(int argc, char *argv[]) {
    int p, q, N = atoi(argv[1]);
    UF_init(N);
    while (scanf("%d %d", &p, &q) == 2) {
        if (!UF_connected(p, q)) {
            UF_union(p, q);
            printf("%d %d\n", p, q);
        }
    }
}
```

- Recebe o número N de objetos como argumento.
- Repete:
 - Lê um par de inteiros de `stdin`.
 - Se eles ainda não estão conectados, faz a união e imprime o par em `stdout`.

Arquivo `client.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include "UF.h"

int main(int argc, char *argv[]) {
    int p, q, N = atoi(argv[1]);
    UF_init(N);
    while (scanf("%d %d", &p, &q) == 2) {
        if (!UF_connected(p, q)) {
            UF_union(p, q);
            printf("%d %d\n", p, q);
        }
    }
}
```

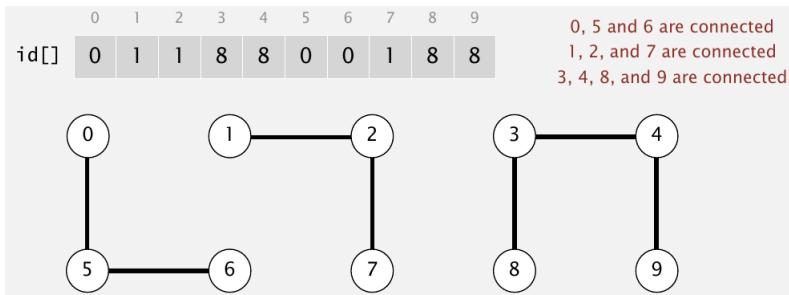
```
$ ./a.out 10
stdin  stdout
4 3    4 3
3 8    3 8
6 5    6 5
9 4    9 4
2 1    2 1
8 9
5 0    5 0
7 2    7 2
6 1    6 1
1 0
6 7
```

Estrutura de dados:

- Um *array* de inteiros `id[]` de tamanho N .
- **Interpretação:** `id[p]` é o identificador da componente conexa que contém p .
- O `id` de um objeto é escolhido **arbitrariamente** para ser o representante da componente aonde ele pertence.

Estrutura de dados:

- Um *array* de inteiros `id[]` de tamanho N .
- **Interpretação:** `id[p]` é o identificador da componente conexa que contém p .
- O `id` de um objeto é escolhido **arbitrariamente** para ser o representante da componente aonde ele pertence.



Operações:

- **Find:** Qual é o `id` de p ? $\Rightarrow \text{id}[p]$.
- **Connected:** Os objetos p e q têm o mesmo `id`?
`id[6] = 0; id[1] = 1;`
 \Rightarrow 6 e 1 não estão conectados
- **Union:** Para unir as componentes contendo p e q ,
modifique todas as posições contendo `id[p]` para `id[q]`.

Quick-find

Operações:

■ **Find:** Qual é o `id` de p ? $\Rightarrow \text{id}[p]$.

■ **Connected:** Os objetos p e q têm o mesmo `id`?

`id[6] = 0; id[1] = 1;`

\Rightarrow 6 e 1 não estão conectados

■ **Union:** Para unir as componentes contendo p e q , modifique todas as posições contendo `id[p]` para `id[q]`.



Ver arquivo `15DemoQuickFind.mov`

Quick-find – Implementação

```
static int id[1000];
static int N;

void UF_init(int size) {
    N = size;
    for (int i = 0; i < N; i++) {
        id[i] = i; // Cada objeto começa na sua propria componente.
    }             // N acessos ao array.
}

int UF_find(int p) {
    return id[p]; // Retorna o id da componente de p. 1 acesso.
}

void UF_union(int p, int q) {
    int pid = id[p];
    int qid = id[q];
    for (int i = 0; i < N; i++) {
        if (id[i] == pid) { // Troca todas os valores de id[p]
            id[i] = qid;    // por id[q].
        }                  // No maximo 2N + 2 acessos ao array.
    }
}
```

Quick-find é muito lento

Modelo de custo: número de acessos ao *array* (leitura ou escrita).

Ordem de crescimento do número de acessos por operação.

Algoritmo	init	union	find	connected
<i>quick-find</i>	N	N	1	1

Quick-find é muito lento

Modelo de custo: número de acessos ao *array* (leitura ou escrita).

Ordem de crescimento do número de acessos por operação.

Algoritmo	init	union	find	connected
<i>quick-find</i>	N	N	1	1

- **União é muito custosa:** operação requer N^2 acessos ao *array* para processar uma sequência de N operações *union* sobre N objetos.
- \Rightarrow Algoritmo **quadrático**.

Algoritmos quadráticos não escalam

Análise simplificada:

- 10^9 operações por segundo.
- 10^9 *words* de memória principal.
- \Rightarrow Acesso a toda memória leva ≈ 1 s.

Algoritmos quadráticos não escalam

Análise simplificada:

- 10^9 operações por segundo.
- 10^9 *words* de memória principal.
- \Rightarrow Acesso a toda memória leva ≈ 1 s.

Grande problema:

- 10^9 operações sobre 10^9 objetos.
- *Quick-find* leva mais de 10^{18} ops.
- 30+ anos de tempo de computação.