

TÓPICOS PARA ORIENTAÇÃO DE ALUNOS

PROF. ROGÉRIO BRITO

Abaixo estão listados **alguns** tópicos para orientação de alunos em Trabalhos de Graduação Interdisciplinar (TGI) ou em Trabalhos de Iniciação Científica.

Observação. O tamanho indicado para alunos não necessariamente é o tamanho ideal para projetos de curto prazo (Trabalhos de Conclusão de Curso). Nesse caso, mais alunos podem ser incluídos no trabalho.

Caso você tenha interesse em algum outro tópico que não esteja listado, entre em contato comigo através do endereço de e-mail rbrito@mackenzie.com.br. Há outros tópicos em que eu tenho interesse em orientar alunos!

1. SOFTWARE LIVRE

1.1. Estudo da Infra-Estrutura do Projeto Debian.

Área: Tecnologia da Informação, Engenharia de Software.

Descrição: Um dos mais bem sucedidos projetos de software livre é o projeto Debian, cujo objetivo é disponibilizar um sistema operacional livre (no sentido mais amplo da palavra, consistindo inclusive de vários aplicativos e *não só* de um kernel).

O projeto Debian é base para diversas distribuições de GNU/Linux utilizadas por grande parte das pessoas expostas a software livre (incluindo distribuições como Knoppix, Kurumin e Ubuntu, dentre inúmeros outros).

O projeto Debian provê software para diversas plataformas, sendo algumas das mais populares (em ordem alfabética) `alpha`, `amd64`, `arm`, `i386`, `m86k`, `powerpc`, `sparc`, dentre várias outras. É interessante notar que, a menos de casos específicos, *todo* software que está disponível para uma plataforma também está para outra plataforma.

O propósito deste trabalho é em compreender como funciona o processo de desenvolvimento e infra-estrutura do Projeto Debian (incluindo a elaboração de pacotes e a forma de inclusão de um pacote na sua base de programas)

Tamanho ideal do grupo: de 2 a 3 alunos.

1.2. Estudo das Políticas de “Fair Queueing” e “Quality of Service” (QoS) em Linux.

Área: Sistemas Operacionais, Tecnologia da Informação.

Date: 2 de março de 2007 (Revisão: 1.4).

Descrição: Com o uso cada vez mais freqüente de redes baseadas em TCP/IP e, com raras exceções, haver banda sem uso para a Internet, um problema surge quando uma nova conexão deve ser feita: o congestionamento da rede freqüentemente causa entraves para conexões que precisam de serem feitas brevemente, mas com alta prioridade.

Essa atribuição de prioridades pode ser feita através de métodos conhecidos como “*Fair Queueing Methods*” e são, ainda hoje, pouco documentados e/ou conhecidos.

O objetivo deste trabalho é estudar o funcionamento de “Fair Queueing” e “Quality of Service” (QoS) em Linux, sua implementação e a interface com o usuário.

Tamanho ideal do grupo: 1 aluno.

2. ANÁLISE E QUALIDADE DE ESCRITA DE SOFTWARE

2.1. Análise de Qualidade de Código em Linguagem C.

Área: Linguagens de Programação, Engenharia de Software, Compilação, Tecnologia da Informação.

Descrição: Uma das tarefas práticas mais difíceis na área de Computação é, de fato, a escrita de software correto e que funcione de acordo com suas especificações.

Enquanto testes de software são insuficientes para garantir a corretude de programas, eles são bastante importantes na detecção de bugs, principalmente se realizados de forma metódica e sistemática.

Testes automáticos de software, entretanto, não são a única ferramenta de que dispomos quando escrevemos software. Além das diversas técnicas para melhor escrita de software, uma classe de ferramentas, as chamadas “Ferramentas de Análises Estáticas” (essencialmente, compiladores que simplesmente verificam o código escrito para um determinado programa) são de grande importância e são utilizados há décadas.

O propósito deste trabalho é estudar e compreender alguns dos analisadores estáticos de código para linguagem C como o “splint” e o “ncc” (vide: <http://students.ceid.upatras.gr/~sxanth/ncc>), dentre outras ferramentas.

Tamanho ideal do grupo: 2 alunos.

2.2. Qualidade, Elegância e Corretude, o Legado de Dijkstra.

Área: Linguagens de Programação, Engenharia de Software, Qualidade de Software, Tecnologia da Informação.

Descrição: Um dos mais influentes Cientistas da Computação foi o *físico* holandês Edsger Wybe Dijkstra, que é conhecido por suas inúmeras contribuições em Computação, variando desde a escrita de software concorrente na década de 1970, quando computadores eram bastante mais raros do que hoje em dia, passando pelo ensino de Ciência da Computação, por lógica (matemática), algoritmos em Teoria dos Grafos (o célebre Algoritmo para rota de menor custo entre dois pontos) e por sua peculiar crítica às formas de escrita de software (que, infelizmente, são preservadas até hoje em dia).

O Prof. Dijkstra deixou, além de uma enorme e influente marca na Computação, uma incrível coleção de escritos que tratam de uma vastíssima quantidade de assuntos, muitas vezes com um humor ácido (necessário para contrabalancear as visões opostas).

O propósito deste trabalho é estudar e compreender alguns dos textos do Prof. Dijkstra, sua visão de como software deve ser elaborado e produzir, de forma concreta, legendas (em inglês e traduzidas para português) de uma entrevista concedida em 2001 (ano anterior a seu falecimento) de um documentário produzido pela rede holandesa VPRO e produzir, paralelamente aos estudos, um texto acessível a alunos de graduação.

Tamanho ideal do grupo: 2 aluno.

3. ESTUDO DE IMPORTANTES ESTRUTURAS DE DADOS MENOS COMUNS

3.1. Árvores de Sufixos.

Área: Linguagens de Programação, Estruturas de Dados

Descrição: Árvores de sufixos são importantes estruturas de dados com inúmeras aplicações em Computação. Suas aplicações vão desde clássicos problemas como o Problema de Casamento de Padrões (“string matching”) até compressão de dados sem perdas (como, por exemplo, os Algoritmos de Lempel e Ziv, bastante populares em utilitários como WinZIP, PKZip etc).

Sua construção em tempo linear requer uma sofisticada análise de complexidade (amortizada). Além disso, interessantes fatos podem ser observados em uma cuidadosa (e reutilizável) implementação.

O propósito deste trabalho é realizar uma implementação (em C, C++ ou Java) e análise de complexidade desta riquíssima estrutura de dados e apresentar algumas de suas inúmeras aplicações.

Tamanho ideal do grupo: 2 alunos.

3.2. Filas de Prioridade.

Área: Linguagens de Programação, Estruturas de Dados

Descrição: Várias são as ocasiões em que estruturas de dados que implementam filas de prioridades são necessárias (isto é, quando há necessidade de atender a demandas por objetos, sendo que cada objeto possui uma certa prioridade fixada). Muitas são as possíveis implementações de filas de prioridades, desde as mais simples (utilizando vetores ingenuamente) até o uso de heaps binários (clássicas estruturas de dados também utilizadas no Algoritmo Heap Sort), Heaps Binomiais e Heaps de Fibonacci (estes últimos, de complexidade assintoticamente menor do que as demais implementações).

O propósito deste trabalho é realizar uma implementação (em C, C++ ou Java) e análise de complexidade destas estruturas de dados e apresentar algumas de suas inúmeras aplicações (como um calendário em que as tarefas ganham maior prioridade conforme a data de seu cumprimento se aproxima).

Tamanho ideal do grupo: 2 aluno.

3.3. Árvores Binárias de Busca e Estruturas Relacionadas.

Área: Linguagens de Programação, Estruturas de Dados

Descrição: Uma das formas mais comuns de manter um dicionário em um computador (isto é, uma estrutura de dados que associe um valor a uma chave, que possivelmente seja não-numérica) é através do uso de árvores binárias de busca.

O tempo utilizado para operações realizadas com árvores é, via de regra, atrelado à profundidade (i.e., quantidade de níveis) de uma árvore de binária de busca. Árvores Binárias de Busca que tenham pequena profundidade (i.e., sejam “balanceadas”) são uma solução normalmente adotada para lidar com degradação de desempenho da estrutura para manter o dicionário. Árvores Balanceadas classicamente implementadas (e estruturas de dados relacionadas) incluem Árvores AVL, Árvores Rubro-Negras, B-árvores, Splay Trees, Treaps e Skip-Lists.

O propósito deste trabalho é realizar uma implementação (em C, C++ ou Java) e análise de complexidade destas estruturas de dados e apresentar algumas de suas inúmeras aplicações (como, por exemplo, aplicações em para armazenamento de dados em Bancos de Dados ou implementação de estruturas como “hashes” em Perl ou “dictionaries” em Python).

Tamanho ideal do grupo: 3 alunos.

3.4. Estruturas de Dados para Conjuntos Disjuntos.

Área: Linguagens de Programação, Estrutura de Dados

Descrição: Em diversas ocasiões, estruturas de dados para manter partições de um conjunto (ou, equivalentemente, uma relação de equivalência) são necessárias. Uma estrutura de dados muito eficiente para este problema é a estrutura proposta por Robert Endre Tarjan, denominada Union-Find. Sua complexidade de tempo amortizada, quando duas heurísticas são utilizadas, é bastante baixa (e, para a esmagadora maioria dos casos práticos, comporta-se de maneira essencialmente linear).

Tamanho ideal do grupo: 1 aluno.

4. ALGORITMOS PARA BUSCA DE PADRÕES EM TEXTOS

4.1. Algoritmos para Buscas de Padrões em Textos.

Área: Projeto e Análise de Algoritmos.

Descrição: Uma das aplicações mais comuns hoje em dia em computação é a busca eficiente de informações a partir de palavras-chave (por exemplo, em sistemas de busca na Internet). A base da busca eficiente de informações é a busca de uma palavra ou pequena frase (normalmente denominado “padrão”) em um texto.

Há diversos algoritmos para realizar a busca de padrões em textos e, além do algoritmo trivial, de tempo quadrático de pior caso, há literalmente dezenas de algoritmos mais sofisticados, dentre os quais os que mais se sobressaem são os Algoritmos de Karp-Rabin, Algoritmo via Autômatos Finitos Determinísticos, o Algoritmo de Boyer-Moore e o Algoritmo de Knuth-Morris-Pratt.

Tamanho ideal do grupo: 1 aluno.

5. TÓPICOS EM OTIMIZAÇÃO COMBINATÓRIA

5.1. Problemas de Caminho de Custo Mínimo.

Área: Logística, Otimização Combinatória.

Descrição: O problema de encontrar a melhor rota entre dois pontos de uma cidade é um problema de grande interesse prático. O critério de otimalidade pode ser especificado de diversas formas, como menor tempo para o traslado, menor distância a ser percorrida, menor número de ruas ou avenidas a serem empregadas etc.

Outro problema de interesse para a própria Computação é a descoberta (ou roteamento) de pacotes de informação em uma rede de computadores.

Além disso, muitos outros problemas podem ser interpretados (ou transformados) como problemas de caminhos de custo mínimo e a forma mais natural de estudar tais problemas é com objetos chamados grafos dirigidos.

Para esses problemas, muitos algoritmos foram desenvolvidos, dependendo de quais sejam as restrições postas para um determinado caso particular (por exemplo, se apenas custos não-negativos são admitidos para um arco do grafo). Há algumas variantes de maior importância para esse problema e tais variantes podem ser resolvidas por clássicos algoritmos em Ciência da Computação, como o Algoritmo de Dijkstra, o Algoritmo de Bellman-Ford, o Algoritmo de Floyd-Warshall e o Algoritmo de Johnson.

Tamanho ideal do grupo: 1 aluno.

5.2. Problema do Fluxo Máximo em Redes.

Área: Logística, Otimização Combinatória.

Descrição: Muitos problemas práticos podem ser vistos como casos particulares de um problema geral em Ciência da Computação, o chamado Problema do Fluxo Máximo em uma rede. Neste problema, uma rede é dada como entrada e o objetivo é encontrar uma forma de distribuir recursos entre os nós da rede de forma que a maior quantidade de tais recursos possa ser enviada de um ponto de origem a um ponto de destino.

Por exemplo, descobrir a melhor forma de escoamento de produção de uma empresa para seus centros consumidores é um problema que pode ser visto como um caso particular do Problema de Fluxo Máximo.

O objetivo deste trabalho é estudar os vários algoritmos existentes para o Problema do Fluxo Máximo em Redes (incluindo os algoritmos baseados no Algoritmo de Ford-Fulkerson e suas variantes e algoritmos do tipo “pre-flow push”).

Tamanho ideal do grupo: 1 aluno.

5.3. Problema do Fluxo Viável de Custo Mínimo em uma rede.

Área: Logística, Otimização Combinatória.

Descrição: Uma generalização importante do Problema do Fluxo Máximo em uma rede é o Problema de Fluxo Viável de Custo Mínimo. Os algoritmos existentes para o problema são de grande importância (dada a relevância do problema, que é bastante geral) e eles podem, em muitos casos, serem interpretados como versões combinatórias de aplicações de métodos de Programação Linear a um determinado sistema de restrições.

O objetivo deste trabalho é estudar os principais algoritmos existentes para o Problema do Fluxo Viável de Custo Mínimo.

Tamanho ideal do grupo: 2 alunos.

5.4. Problemas de Emparelhamentos em Grafos.

Área: Logística, Otimização Combinatória.

Descrição: Muitos problemas práticos podem ser modelados como problemas em grafos e um clássico problema a ser resolvido é o problema de, dados n homens e n mulheres, sendo que cada par possui um certo “grau de afinidade”, fazer o papel de uma agência matrimonial e conseguir realizar o maior número possível de casamentos (ou, alternativamente, encontrar uma forma de casar o maior número de pessoas de forma que as afinidades entre os pares sejam respeitadas tanto quanto possível).

Esse problema, formulado na linguagem de Teoria dos Grafos, consiste em encontrar um Emparelhamento de Custo Máximo em um grafo bipartido com custos nas arestas.

O objetivo deste trabalho é estudar algoritmos eficientes para encontrar emparelhamentos em grafos (não necessariamente bipartidos).

Tamanho ideal do grupo: 2 alunos.

5.5. Estudo Introdutório sobre Matróides.

Área: Logística, Otimização Combinatória.

Descrição: O objetivo deste trabalho é estudar estruturas algébricas importantes conhecidas como matróides que estão por trás de vários algoritmos eficientes (algoritmos gulosos) frequentemente encontrados em Logística.

Tamanho ideal do grupo: 1 aluno.

5.6. Estudo do Método Primal-Dual.

Área: Logística, Otimização Combinatória.

Descrição: Estudo da forma geral do Método Primal-Dual para resolução de problemas em Otimização Combinatória e estudo de interpretação de algoritmos clássicos como casos especiais do Método Primal-Dual.

Tamanho ideal do grupo: 1 aluno.

5.7. Estudo de Algoritmos de Aproximação Combinatórios.

Área: Logística, Otimização Combinatória, Análise de Algoritmos.

Descrição: Algoritmos de aproximação para problemas NP-difíceis como “Vertex Cover” (2-aproximações), “Escalonamento de Tarefas em Máquinas Idênticas” (Algoritmo de Graham), Problema “Set Cover” (Algoritmo de Chvátal), dentre outros, Problemas de empacotamento (“Bin Packing”), Problema da Mochila (“The Knapsack Problem” e Algoritmo de Ibarra-Kim). Classes de Aproximação (FPTAS, PTAS, APX) e inaproximabilidade de Problemas (e.g., Problema do Caixeiro Viajante).

O objetivo deste trabalho é estudar a teoria geral de algoritmos de aproximação e exibir alguns casos clássicos em que eles são empregados, bem como mostrar sua classificação. É objetivo também implementá-los (em C, C++ ou Java).

Tamanho ideal do grupo: 3 alunos.

5.8. Estudo do Método Simplex.

Área: Logística, Otimização Combinatória, Análise de Algoritmos, Programação Matemática.

Descrição: Forma básica do Método Simplex. Regras para convergência (Regra de Bland). Implementação do Algoritmo Simplex. Simplex Revisado. Famílias de casos “patológicos” em que o Algoritmo Simplex pode requerer um número exponencial de passos.

Tamanho ideal do grupo: 2 alunos.

5.9. Algoritmo Simplex para Redes.

Área: Logística, Otimização Combinatória, Análise de Algoritmos, Programação Matemática.

Descrição: Uma versão especializada do Algoritmo Simplex quando usado para problemas em redes (isto é, grafos dirigidos), que permite uma interpretação combinatória das operações de pivotação e de folgas-complementares.

Tamanho ideal do grupo: 1 aluno.

6. TÓPICOS EM COMPUTABILIDADE E COMPLEXIDADE COMPUTACIONAL

6.1. Problemas Clássicos de Computabilidade.

Área: Teoria da Computação, Computabilidade, Complexidade Computacional.

Descrição: Muitos programadores estão acostumados com o fato de que, quando eles escrevem um código em alguma linguagem (digamos, C), o compilador pode efetuar análises do texto escrito pelo programador e fazer alterações que preservem o comportamento do programa, mas que ofereça alguma vantagem (por exemplo, menor consumo de memória ou menor consumo de tempo). Esta atividade realizada por compiladores é chamada comumente “otimização de código”.

Além disso, um compilador moderadamente sofisticado pode analisar o fluxo de execução do código-fonte escrito pelo programador e indicar erros comuns como, por exemplo, se há variáveis que tenham sido definidas, mas que não tenham sido utilizadas (e que, portanto, são desnecessárias para o programa) ou variáveis que tenham sido definidas, mas não tenham sido inicializadas (e que, muito possivelmente, trarão comportamento indefinido para o programa). Outra tarefa que um compilador pode fazer é, em alguns casos, detectar código escrito pelo programador que nunca poderá ser executado (por exemplo, se a execução do código depender de alguma condição que o compilador veja que nunca poderá ser satisfeita).

É natural, em vista desse processo de “otimização de código”, que nós nos perguntemos se um compilador não poderia oferecer a conveniência de, durante o processo de compilação, determinar se um programa irá ou não ficar em um “laço infinito” (isto é, nunca ter sua execução concluída). Este é o clássico problema chamado “Problema da Parada” e, infelizmente, é possível provar que *não* existe um compilador (ou interpretador) que consiga decidir *sempre* se um dado programa vai ou não entrar em um “laço infinito”, nem mesmo se nós fornecermos, antecipadamente, a entrada que o programa utilizará para suas computações.

É o objetivo deste trabalho estudar o Problema da Parada e problemas similares, como os Teoremas de Gödel (Teoremas da Completude e da Incompletude).

Tamanho ideal do grupo: 1 aluno.

6.2. Demonstração de que PRIMOS está em NP.

Área: Teoria da Computação, Complexidade Computacional.

Descrição: Demonstração de Vaughan Pratt de que o problema de decidir se um número é primo é um problema que está na classe NP de problemas.

Tamanho ideal do grupo: 1 aluno.

7. TEORIA DA INFORMAÇÃO

7.1. Análise de Compressores (sem perdas) de Áudio.

Teoria da Informação.

Descrição: Em várias situações de nosso dia-a-dia, nós nos deparamos com a tarefa de armazenamento de informações. Um tipo de informações que requer bastante atenção, devido à quantidade de dados utilizada é a Compressão de Dados em forma de som.

A compressão de dados pode ser *com perdas* (quando a versão restaurada não é igual à versão comprimida) ou pode ser **sem perdas** (quando o que é descomprimido é uma versão idêntica àquilo que foi comprimido).

Apesar de a modalidade com perdas (MP3, AAC, WMA, MPC, Vorbis, Speex) atingir maiores taxas de compressão, ela sacrifica (às vezes, muito) da quantidade de dados que queremos manter, para compensar tal perda em pequeno consumo de espaço.

Por outro lado, compressores de áudio que são sem perdas, são, tipicamente, mais simples de serem codificados e a justificativa para isso é o fato de que eles raramente empregam modelos psico-acústicos ou mascaramento de ruídos para eliminação de informações.

O propósito deste trabalho é em compreender como funcionam sistemas de compressão de áudio que trabalhem sem perdas, tendo, como base, o compressor FLAC (Free Lossless Audio Codec) e seu predecessor, o Shorten (muito utilizado por admiradores de música que não querem perder qualidade de gravações raras de áudio e para mantê-las arquivadas). É também o propósito deste trabalho produzir, paralelamente aos estudos, um texto acessível a alunos de graduação.

7.2. Compressão de Dados via Algoritmos Não-Estatísticos.

Área: Teoria da Informação.

Descrição: Algoritmos LZ77, LZ78, LZW e variantes.

Estudar as bases, implementar cuidadosamente (em C, C++ ou Java) e fazer um estudo empírico de como os clássicos algoritmos derivados das propostas de Lempel e Ziv (por exemplo, LZ77, LZ78, LZW, LZFG etc) comportam-se com dados “realísticos”.

Tamanho ideal do grupo: 2 alunos.

7.3. Compressão de Dados via Algoritmos Estatísticos.

Área: Teoria da Informação.

Descrição: Estudo de uma das formas mais simples e eficazes (Codificação de Huffman, um algoritmo guloso) de codificações sem perdas, utilizada em uma variedade imensa de projetos de software como, por exemplo, codificadores de MP3). Versão mais geral do que Codificação de Huffman): codificação aritmética. Modelagem estatística para os geradores de códigos citados.

O objetivo deste projeto é compreender as diversas formas de codificação eficientes de dados (algumas, inclusive para compressão de textos) e produzir, paralelamente aos estudos, um texto acessível a alunos de graduação.

Tamanho ideal do grupo: 2 alunos.

7.4. Algoritmo de Burrows-Wheeler.

Área: Teoria da Informação.

Descrição: Algoritmo não-estatístico (diferentemente do algoritmo de Huffman) com taxa de compressão consideravelmente melhor do que os Algoritmos baseados nos métodos LZ77, LZ78 e variantes. É implementado no utilitário bzip2, amplamente distribuído em sistemas UNIX. Este algoritmo também é conhecido como algoritmo de “block sorting”.

O objetivo deste projeto é compreender o funcionamento do Algoritmo de Burrows-Wheeler e produzir, paralelamente aos estudos, um texto acessível a alunos de graduação.

Tamanho ideal do grupo: 1 aluno.

7.5. Algoritmo LZMA.

Área: Teoria da Informação.

Descrição: O recente Algoritmo LZMA é uma variante dos algoritmos da família de Lempel e Ziv (algoritmos baseados em argumentos simples, sem necessidade de “background” em probabilidade) e que, freqüentemente, atinge taxas de compressão superiores aos outros algoritmos não-estatísticos. O Algoritmo LZMA é implementado no popular utilitário 7zip.

O objetivo deste projeto é compreender o funcionamento do Algoritmo LZMA e produzir, paralelamente aos estudos, um texto acessível a alunos de graduação.

Tamanho ideal do grupo: 1 aluno.

7.6. Códigos Detectores e Códigos Corretores de Erros.

Área: Teoria da Informação.

Descrição: Códigos de Hamming, Códigos Perfeitos, Códigos de Goppa, Códigos BCH, Códigos de Reed-Solomon. Aplicações em codificação de CDs e DVDs.

Tamanho ideal do grupo: 3 alunos.

8. TEORIA DOS NÚMEROS ALGORÍTMICA

8.1. Algoritmos para Cálculo de MDC.

Área: Segurança de Redes, Teoria dos Números, Análise de Algoritmos.

Descrição: Há diversos algoritmos de grande importância em Teoria dos Números. Um dos algoritmos mais célebres, robustos e bem estudados é o Algoritmo de Euclides para cálculo do *máximo divisor comum* entre dois números. Versões estendidas desse algoritmo nos conduzem a algoritmos para, dentre outras coisas, calcular inversos de elementos módulo um número inteiro (i.e., cálculo de inversos em (\mathbb{Z}_n) , para algum inteiro $n \geq 2$).

Além disso, o Algoritmo de Euclides é bastante eficiente do ponto de vista computacional: seu consumo de tempo é polinomial no tamanho da entrada do problema. Este fato, que não é imediatamente óbvio, é conhecido como *Teorema de Lamé* e, curiosamente, na análise de tempo do Algoritmo de Euclides, os números de Fibonacci são empregados.

O objetivo deste trabalho é estudar assuntos elementares de Teoria dos Números Algorítmica para realizar um estudo de algoritmos para cálculo do MDC de dois números.

Tamanho ideal do grupo: 1 aluno.

8.2. Algoritmos para Teste de Primalidade.

Área: Teoria dos Números, Análise de Algoritmos.

Descrição: Números de Carmichael, números pseudo-primos, Algoritmo de Miller-Rabin.

Tamanho ideal do grupo: 1 aluno.

8.3. Algoritmo RSA de Criptografia de Chave Pública.

Área: Segurança de Redes, Teoria dos Números, Criptografia.

Descrição: Algoritmos de Chave Pública. Algoritmo RSA.

O objetivo deste trabalho é estudar assuntos elementares de Teoria dos Números Algorítmica para realizar um estudo do algoritmo RSA de Criptografia de Chave Pública, usado em conexões seguras em navegadores convencionais (e.g., Firefox, Internet Explorer, Safari).

Tamanho ideal do grupo: 1 aluno.

8.4. Prova de que PRIMOS está em NP.

8.5. Algoritmo de A-K-S.

Área: Teoria dos Números, Teoria Algébrica dos Números, Teoria Algorítmica dos Números.

Descrição: Recente demonstração (de 2002/2003) de que o problema de decidir se um dado número positivo natural p é um número primo é um problema que está na classe P de algoritmos (já era conhecido que $\text{PRIMOS} \in \text{NP} \cap \text{co-NP}$).

Tamanho ideal do grupo: 2 alunos.

9. TÓPICOS EM TEORIA DOS GRAFOS

9.1. Problemas Clássicos de Teoria dos Grafos.

Área: Teoria dos Grafos, Matemática Discreta.

Descrição: Demonstrações de teoremas clássicos como Teorema de Kuratowski, Teorema das Seis Cores, Teorema das Cinco Cores, noção do Teorema das Quatro Cores. Teorema (extremal) de Turán, versões simplificadas do problema (para grafos sem triângulos) e várias demonstrações, inclusive para o caso geral.

Tamanho ideal do grupo: 1 aluno.

9.2. Teoria dos Grafos Algébrica.

Área: Teoria dos Grafos, Álgebra Abstrata.

Descrição: Aplicações de Álgebra Linear e Álgebra Abstrata à Teoria dos Grafos.

Tamanho ideal do grupo: 2 alunos.

9.3. Teoria dos Grafos Topológica.

Área: Teoria dos Grafos, Topologia.

Descrição: Generalizações do Problema da Planaridade de Grafos.

Tamanho ideal do grupo: 2 alunos.