Osório Cabo Winter

Relações de Recorrência: Para Além de P.A. e P.G.

Santo André, 2013

Universidade Federal do ABC

Centro de Matemática, Computação e Cognição

Osório Cabo Winter

Relações de Recorrência: Para Além de P.A. e P.G.

Orientador: Prof. Dr. Rafael de Mattos Grisi

Dissertação de mestrado apresentada ao Centro de Matemática, Computação e Cognição para obtenção do título de Mestre em Matemática

Este exemplar corresponde a versão final da dissertação
defendida pelo aluno Osório Cabo Winter,
e orientada pelo Prof. Dr. Rafael de Mattos Grisi.

Santo André, 2013

Folha de Aprovação

Substituir pelo pdf da folha de aprovação assinada pela banca.

AGRADECIMENTOS

Agradeço a DEUS, que sempre me iluminou, além de ter me dado condições de concluir mais esta etapa da minha vida com saúde, paz e sabedoria.

A minha mãe, Olga, que sempre me apoiou e acreditou no meu trabalho.

Aos meus filhos, Gustavo, Gabriel e Gabriela que souberam suportar esse período de dedicação e entrega, sem poder passear, viajar nas férias, sem ter a minha presença e a atenção que tanto mereciam.

À minha esposa, Berenice, que, com muita paciência e sabedoria dispensou-me a sua total dedicação e apoio em todos os momentos tendo, ainda, a sensibilidade de compensar a minha ausência junto aos meus filhos.

Aos professores do PROFMAT-UFABC, que nos incentivaram, com dedicação, a suprir nossas deficiências e alcançar nosso objetivo, auxiliando-nos no que foi preciso.

Em especial, ao meu orientador Prof. Dr. Rafael de Mattos Grisi, pela excepcional paciência, ajuda, orientação e empenho no desenvolvimento e sem o qual não existiria este trabalho.

Aos idealizadores e a CAPES, pois se propuseram e alcançaram a formação dessa primeira turma, sendo que foi fundamental o apoio financeiro para cada um dos mestrandos.

RESUMO

Este trabalho tem a finalidade de mostrar que o ensino de equações de recorrência no ensino médio pode ir além do tradicional estudo das progressões aritméticas (P.A.) e progressões geométricas (P.G.), partindo da apresentação de problemas motivadores e encerrando cada capítulo com atividades propostas para verificação. Para isso estudaremos alguns problemas clássicos, cuja solução passa pelo estudo de relações de recorrência. Dentre eles estudares o problema das Torres de Hanoi , o problema de Josephus e o da procriação de coelhos. Mostraremos técnicas para estudar e resolver alguns tipos de equações e problemas de valor inicial. Para terminar exploraremos a conexão entre computação e equações de recorrência no estudo do tempo de execução de algoritmos recursivos, em especial algoritmos de ordenamento e busca em listas.

Palavras-chave: Recorrência, Algoritmos, Complexidade

ABSTRACT

This work aims to show that the teaching of recurrence equations in high school can go far beyond the traditional study of arithmetic progressions and geometric progressions, based on the presentation of motivating problems and ending each chapter with proposed activities for verification. In order to do that, we'll study some classical problems whose solution involves recurrence relations. Among them are the problem of the Towers of Hanoi, the problem of Josephus and the breeding of rabbits. We will show techniques to study and solve some types of recurrence equations and initial value problems. To finish we will explore the connection between computation and recurrence equations in the study of the running time of recursive algorithms, especially algorithms for sorting and searching.

Keywords: Recurrence, algorithm, complexity

CONTEÚDO

INTRODUÇÃO

Ao iniciar o estudo com a proposta de problemas tradicionais ou que façam referência a algo que pode interessar aos alunos, normalmente, levam os mesmos a refletirem sobre os problemas apresentados e assim despertar interesse pelos assuntos que devem surgir e serem apresentados logo em seguida aos exercícios.

A apresentação do conteúdo, leva os alunos a tentarem associar e utilizar aquilo que está sendo trabalhado na resolução dos problemas que estão aguardando para serem resolvidos.

Um algoritmo é um procedimento consistindo de um conjunto de regras, provido de clareza, as quais devem ser cumpridas em uma sequência finita de operações.

Uma ferramenta útil e poderosa, não só na matemática, mas também para a computação, é a recursividade e um algoritmo recursivo pode ser explorado através das relações dele gerado e que podem ser utilizados na resolução dos problemas aqui propostos .

Vamos apresentar uma forma de trabalhar sobre o assunto proposto com os alunos do ensino médio.

Iniciamos o capítulo 1 apresentando alguns problemas para motivar e despertar o interesse pela resolução dos mesmos. Utilizamos problemas tradicionais como "as torres de Hanoi", "O problemas dos coelhos", onde aparece a conhecida sequência de Fibonacci, e "o problema de Josephus". Em seguida, apresentamos a definição de recursão e alguns trabalhos que mostram o chamado efeito droste de autoria de Maurits Cornelis Escher , um artista gráfico holandês, conhecido pelas suas xilogravuras, litografias e meios-tons, que tendem a representar construções impossíveis . Os fractais que tiveram impulso no seu estudo, e que receberam essa denominação na década de setenta. Depois apresentamos a solução dos problemas iniciais, de forma prática, em ordem do grau de dificuldade e acrescentando-se aos problemas o contexto baseado nos relatos da história da matemática, encerrando o capítulo com algumas atividades propostas visando aprofundamento no assunto tratado.

No segundo capítulo, são apresentados os conceitos teóricos relacionados equações de recorrência. Apresentaremos algumas técnicas de resolução para problemas de valor inicial, apresentando a solução geral para os mesmos, além de outros exemplos de aplicação

No terceiro capítulo tratamos de uma aplicação prática das relações de recorrência: a análise de complexidade de algoritmos. Estudaremos o tempo de execução de diversos algoritmos, analisando sua ordem de grandeza. Para exemplificar, estudamos alguns métodos de ordenação de listas (Selection Sort, Bubble Sort e Merge Sort) passando ao estudo de métodos de busca (Busca Sequencial e Busca Binária).

1

RECURSIVIDADE E RELAÇÕES DE RECORRÊNCIA

"De que me irei ocupar no céu, durante toda a Eternidade, se não me derem uma infinidade de problemas de Matemática para resolver? "

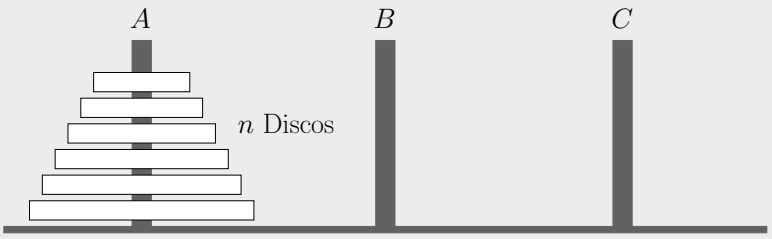
— Augustin Louis Cauchy

1.1 PROBLEMAS RECURSIVOS

Considere os problemas abaixo.

Problema 1. Torres de Hanói

É um jogo que consiste em uma base de madeira onde estão firmados três hastes verticais, e um certo número n de discos de madeira, de diâmetros diferentes, furados no centro e podemos chamar de A, B e C, as três hastes, conforme a figura abaixo.



No começo do jogo os discos estão todos na haste A, em ordem decrescente de tamanho, com o menor disco sobre os demais. O objetivo é mover todos os discos, de A para C, podendo utilizar a haste B, deixando-os na mesma ordem e obedecendo às seguintes regras:

1. Somente um disco pode ser posto de cada vez;
2. Um disco maior nunca pode ser posto sobre um disco menor.

Assim, responda: Qual o número de movimentos necessários para se cumprir essa tarefa se o número de discos é cinco?

Problema 2. O Problema de Josephus (Roleta Romana)

Um grupo de soldados é circundado por uma força inimiga esmagadora. Não há esperança de vitória e só existe um cavalo disponível para escapar! Os soldados entram num acordo para determinar qual deles deverá escapar para trazer ajuda. Para tanto eles irão utilizar o método da roleta romana para selecionar o soldado sortudo. O procedimento funciona da seguinte forma:

1. Os soldados se dispõe em círculo respeitando certa sequência fixa.
2. Um número $n > 0$ é sorteado. Um dos nomes dos soldados também é sorteado.
3. Começando-se pelo soldado cujo nome foi sorteado, eles iniciam uma contagem sequencial ao longo do círculo em sentido horário. Quando a contagem alcança o n-ésimo soldado, ele é retirado do círculo (portanto eliminado da escolha) e a contagem reinicia com o soldado seguinte.
4. O processo continua de maneira que, toda vez que n é alcançado, outro soldado é eliminado do círculo (lembre-se, todo soldado retirado do círculo não entra mais na contagem).
5. O último soldado que restar deverá montar o cavalo e escapar.

Pergunta-se: Em qual posição do circulo deve ficar o soldado para que possa escapar da força inimiga?

Problema 3. O Problema dos Coelhos

Um sitiante resolve criar coelhos e para iniciar essa criação adquire um casal de coelhos recém nascidos e recebe as seguintes instruções:

1. No primeiro mês tem-se apenas um casal;
2. Casais reproduzem-se somente após o segundo mês de vida;
3. Não há problemas genéticos no cruzamento consanguíneo;
4. Todos os meses, cada casal fértil dá à luz um novo casal;
5. Os coelhos nunca morrem.

Pergunta-se: Quantos casais de coelhos podem ser gerados a partir de um casal de coelhos em um ano?

Problema 4. Ordenando Listas

Seja a lista não ordenada:

$$\{23, 27, 41, 19, 91, 62, 54, 26, 38, 12, 10, 11\}.$$

Pergunta-se: Quantas comparações, no mínimo, devem ser feitas entre os elementos da lista dada de modo a deixá-la na forma ordenada crescente?

Problema 5. Busca em Listas

Vamos considerar o seguinte jogo: um jogador A pensa em um número de 1 a 1000, e um jogador B tem que acertar este número "chutando" o menor número de vezes possível. A cada chute do jogador B, o jogador A vai dizer apenas se o número escolhido por ele é maior ou menor que o número dito por B.

Pergunta-se:

Qual o número máximo de chutes que o jogador B precisará dar para descobrir o número escolhido por A? Qual o estratégia que ele deve utilizar para isso?

Apesar de a primeira vista os problemas acima parecerem não relacionados, suas soluções são exemplos de uma importante classe de problemas matemáticos: os problemas recursivos.

Um objeto é denominado recursivo quando sua definição é parcialmente feita em termos dele mesmo. A recursividade (ou recursão) é encontrada principalmente na matemática, mas está presente em diversas situações do cotidiano. Por exemplo, quando um objeto é colocado entre dois espelhos planos, apontados um para o outro, surge uma imagem recursiva (uma imagem do objeto refletida do outro espelho) e sucessivamente surge a mesma figura refletida mais de uma vez.

Outro exemplo, de recursividade, é quando uma figura apresenta a si mesma na própria figura, como na Figura 1.

Figura 1: Forma visual de recursão conhecida como efeito Droste.

Na matemática e na ciência da computação, a recursão constrói uma classe de objetos ou métodos definindo alguns casos base ou métodos simples, para então definir regras que descrevam casos complexos em termos de casos mais simples.

A definição formal dos números naturais, por exemplo, diz que zero é um número natural, e todo número natural tem um sucessor, que é também um número natural. Neste caso, um número natural é definido como sucessor de outro, que é definido da mesma forma, seguindo assim até atingirmos o zero, que foi definido a princípio.

Ou seja, o 51 é sucessor do 50, que por sua vez é sucessor do 49, e assim por diante até chegar ao zero, que é o primeiro elemento definido.

Definições como essa, frequentemente encontradas na matemática, são exemplos de definições recursivas, pois um objeto é definido a partir de um objeto anterior.

Tais problemas possuem portanto a seguinte propriedade: cada *instância* do problema contém uma instância menor do mesmo problema. Dizemos assim que estes problemas possuem estrutura recursiva, e para resolvê-los podemos aplicar o seguinte método:

- se a instância em questão é pequena, resolva-a diretamente;
- senão, reduza-a a uma instância menor do mesmo problema, e aplique o método à instância menor.

Exemplo 1.1. Permutação

A recursividade pode ser utilizada para gerar todas as possíveis permutações de um conjunto de símbolos. Por exemplo, existem seis permutações no conjunto de símbolos A, B e C: ABC, ACB, BAC, BCA, CBA e CAB. O conjunto de permutações de n símbolos é gerado tomando-se cada símbolo por vez e prefixando-o a todas as permutações que resultam dos $(n-1)$ símbolos restantes. Tais passos podem ser esquematizados da seguinte forma.

- Se $n = 1$, então só existe uma permutação possível.
- Se $n > 1$ siga os passos abaixo para cada elemento do conjunto:
  - Retire o elemento escolhido;
  - Gere todas as permutações possíveis dos $(n-1)$ elementos restantes;
  - Junte o elemento escolhido a cada permutação gerada.

Na computação, a aplicação de tal método gera os chamados *algoritmos recursivos*. Dentre os exemplos importantes do uso da recursividade em ciência da computação está um método comum de simplificação que consiste em dividir um problema em subproblemas do mesmo tipo, denominada de *divisão e conquista*. No capítulo 3 trataremos de uma solução utilizando tal método.

Outro conjunto importante de objetos que podem ser definidos recursivamente são os fractais, onde versões menores de um mesmo objeto são recursivamente "colados" no objeto anterior, formando a figura que conhecemos por fractal. As figuras 2 e 3 mostram estágios intermediários da construção de fractais.

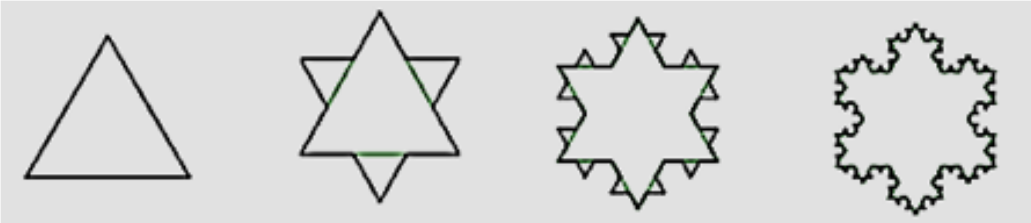


Figura 2: Quatro fases na construção de um Floco de neve de Koch, como em muitos outros fractais, os estágios são obtidos através de uma definição recursiva.

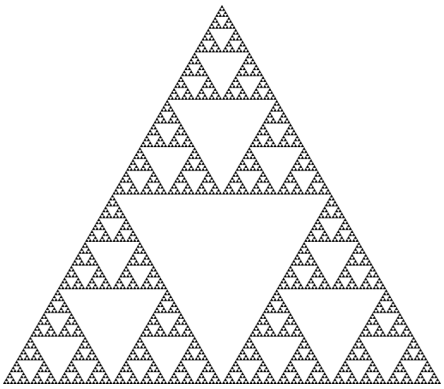


Figura 3: O Triângulo de Sierpinski é formado dividindo um triângulo em 4 triângulos de mesmo tamanho e retirando o triângulo central. Este procedimento é repetido indefinidamente para cada triângulo restante.

A próxima seção é lúdica e será dedicada a um artista que usou a matemática como fonte de inspiração para muitas de suas obras, e onde elementos recursivos são frequentemente encontrados.

1.1.1 *M. C. Escher*

Maurits Cornelis Escher foi um artista holandês do início do século XX, que ficou conhecido pela forte presença de elementos matemáticos em suas obras. Nascido

em Leenward, Holanda em junho de 1898, era filho de um engenheiro civil, que o encorajou desde cedo a aprender artes ligadas à carpintaria. Neste período aprendeu técnicas que o ajudariam muito nos seus trabalhos futuros.

Escher era considerado um aluno fraco na escola, mas seus interesses por gravuras apareceram ainda cedo. Durante seus anos na escola secundária, que nunca concluiu, Escher começou a fazer lineo-gravuras, com a companhia de um amigo.O seu primeiro trabalho, Pássaro numa gaiola, de 1916, não fez sucesso entre seus professores.

O fascínio de Escher pela matemática fica clara em parte significativa de suas obras. Conceitos como rotações, translações, simetrias e até mesmo o infinito, estão presentes em vários de seus trabalhos. É interessante observar no entanto que Escher não tinha nenhuma formação matemática, tendo ele mesmo dito diversas vezes que não se considerava um matemático. Mas não deixava de reconhecer a proximidade de seu trabalho com a matemática.

Seus trabalhos chegaram a ser expostos em uma edição do ICM (International Congress of Mathematicians), onde travou contato com matemáticos famosos como Penrose e Coxeter. Este último chegou a usar ilustrações de Escher em um de seus artigos científicos. Maiores detalhes podem ser encontrados em [2].

Durante sua vida, fez 448 litografias, xilogravuras e gravuras em madeira e mais de 2000 desenhos e esboços. [3]

A seguir, apresentamos três de seus trabalhos.

O primeiro (Figura 4), conhecido como *Fractal do Lagarto*, traz uma estrutura recursiva bastante clara. Cada "anel" de lagartos é repitido sistematicamente para dentro, reduzindo sua área. De acordo com [4], "...cada elemento (em forma de réptil) deste padrão, dirigindo-se para o centro, é sistemática e continuamente reduzida a metade...". Tal padrão pode ser repetido até ficar infinitamente pequeno, dependo principalmente da habilidade do gravador, e da qualidade e precisão dos materiais disponíveis. No caso desta figura, retirada de [4], a precisão é extraordinária. O menor dos répteis tem 2mm, e está desenhado inteiramente, com pernas, cabeça e cauda.

Escher se deteve no estudo dos contrastes e paradoxos visuais, os *loopings eternos*, expondo padrões que se replicam e se refletem ao infinito (antecipando o que veio a ser a geometria fractal de Mandelbrot de 1970).

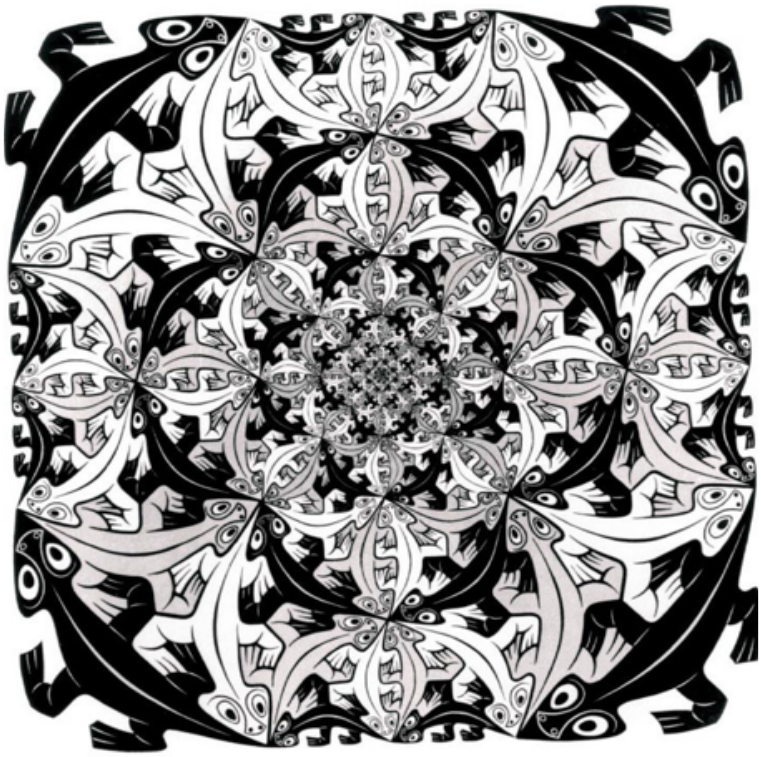


Figura 4: A gravura em madeira: Cada vez mais pequeno I (1956) - M.C.Escher

Na segunda (Figura 5), temos: "As componentes reduzem-se de dentro para fora. As seis maiores, três anjos brancos e três demónios pretos, estão ordenadas radialmente em volta do centro. O disco está dividido em seis setores, onde dominam os anjos, frente a um fundo preto e os demónios, frente a um branco. O céu e o inferno aparecem alternadamente seis vezes.". [3]

A terceira (Figura **??** mostra a Escher desenhando a si mesmo, em uma simples, mas bonita alusão ao conceito de recorrência.

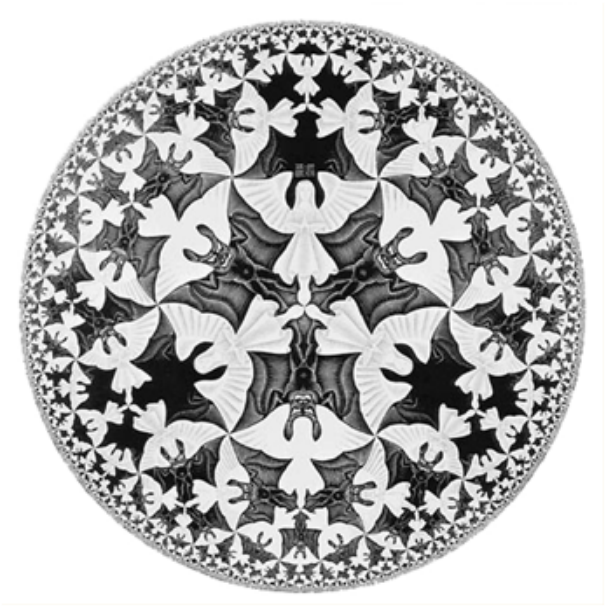


Figura 5: Limite Circular IV (1960) - M.C.Escher

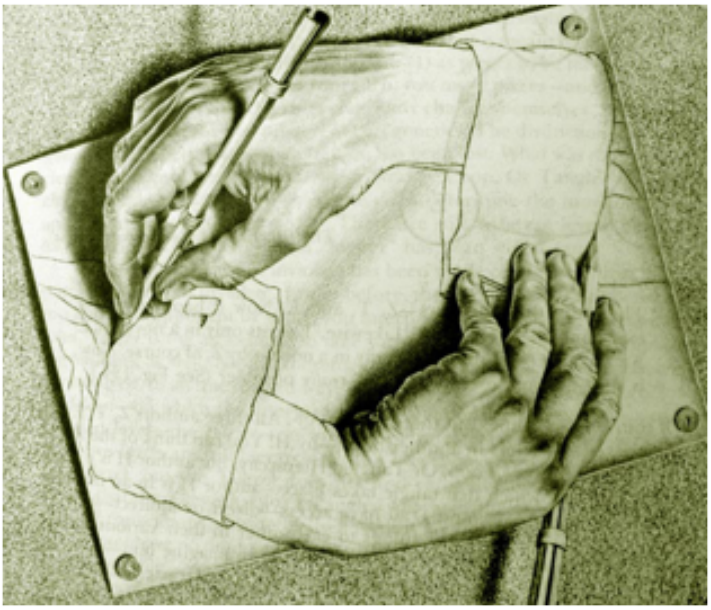


Figura 6: Desenhando-se (1948) - M.C.Escher

1.2 RELAÇÃO DE RECORRÊNCIA

Uma relação de recorrência para a sequência (a_n) é uma equação que expressa a_n em função de um ou mais termos anteriores $(a_1, \ldots, a_{n-1})$, $n = 2, 3, 4, 5 \ldots$, assim uma

sequência é uma solução de uma relação de recorrência se os seus termos satisfazem esta relação de recorrência e as condições iniciais.

As condições iniciais para uma sequência especificam os termos que precedem o primeiro termo onde a relação de recorrência atua.

Exemplo 1.2. Função Fatorial

Esta função é um dos exemplos clássicos de recursividade e, por isso, de citação quase obrigatória. Eis sua definição recursiva:

$$n! = \begin{cases} 1 & , \text{ se } n = 0; \\ n \cdot (n-1)! & , \text{ se } n > 0. \end{cases}$$

Outra forma de apresentar é, fazendo $F(n) = n!$, temos que $F(n)$ pode ser definida por

$$F(n) = \begin{cases} 1 & , \text{ se } n = 0; \\ n \cdot F(n-1) & , \text{ se } n > 0. \end{cases}$$

Assim, podemos verificar que

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1,$$

onde n é um número natural.

Seguindo a definição acima, vamos determinar, por exemplo, o cálculo de 4!.

$$\begin{aligned} 4! &= 4.3! \\ &= 4.[3.(3-1)!\,] \\ &= 4.3.2! \\ &= 4.3.[2.(2-1)!\,] \\ &= 4.3.2.1! \\ &= 4.3.2.1.(1-1)! \\ &= 4.3.2.1.0! \\ &= 4.3.2.1.1 \\ &= 24. \end{aligned}$$

Dois exemplos bastante conhecidos de relações de recorrência presentes no currículo atual do ensino médio são as chamadas progressões aritmética e geométrica.

Exemplo 1.3. Progressão Aritmética - P.A.

Entre as sequências numéricas, se destacam aquelas conhecidas como progressões aritméticas (P.A.), onde cada termo (a partir do segundo) desse tipo de sequência é determinado pela soma do termo anterior com um valor real (r) chamado de razão

$$r = a_{n+1} - a_n, \ n = 1, 2, 3, 4, \ldots$$

Assim a sequência a_n é dada por

$$\begin{aligned} a_2 &= a_1 + r, \\ a_3 &= a_2 + r, \\ a_4 &= a_3 + r, \\ &\vdots \\ a_n &= a_{n-1} + r, \end{aligned}$$

onde a_1 é um valor dado (condição inicial), sendo que a fórmula explicita (ou solução fechada) é:

$$a_n = a_1 + (n-1)r.$$

Exemplo 1.4. Progressão Geométrica - P.G.

Outro grupo de sequências, chamado de progressão geométrica (P.G.), onde cada elemento, a partir do segundo, é determinado pelo produto do termo anterior com

uma constante real, chamada de razão $q = a_{n+1}/a_n$, para $n = 1, 2, 3, 4, \ldots$, e quando não se trata de P.G. quase nula ($q = 0$). Assim a sequência (a_n) é dada por

$$\begin{aligned} a_2 &= a_1 \cdot q, \\ a_3 &= a_2 \cdot q, \\ a_4 &= a_3 \cdot q, \\ &\vdots \\ a_n &= a_{n-1} \cdot q, \end{aligned}$$

onde a_1 é um valor dado (condição inicial), sendo que a fórmula explicita (ou solução fechada) é:

$$a_n = a_1 \cdot q^{n-1}.$$

1.3 RELAÇÕES DE RECORRÊNCIA PARA ALGUNS DOS PROBLEMAS INICIAIS

A seguir, falaremos dos problemas das Torres de Hanoi e de Josephus. O problema de ordenação dos elementos da lista (Problema 4) e o problema de procura de um número em uma lista dada (Problema 5) serão tratados no terceiro capítulo devido a necessidade de verificação dos diferentes métodos de ordenação e de busca que podem ser utilizados para a resolução dos mesmos e que naquele capítulo serão abordados.

1.3.1 *O problema das Torres de Hanoi*

O problema das Torres de Hanói vem servindo de exemplo no estudo de recorrências há muito tempo. Considerações sobre ele foram feitas pelo matemático francês François Édouard Anatole Lucas (1842-1891), professor no Lycée Saint Louis e posteriormente no Lycée Charlemagne, ambos em Paris. Édouard Lucas tratou das Torres de Hanóiem suas notas *Récréations mathématiques - quatrième volume* de 1883, e posteriormente de forma mais elaborada, em 1889, no livro *Jeux Scientifiques pour servir*

à lt'Histoire, à lÉnseignement et à la Pratique du Calcule et du Dessin - Premiére sèrie n^o 3 [5].

De acordo com [6], o problema Torres de Hanói tem sua origem em uma antiga lenda hindu, passada na cidade sagrada de Bernares na Índia. Conta a lenda que em Bernares existia um tempo com 3 torres sagradas do bramanismo, e que no início dos tempos os monges teriam recebido uma pilha de 64 discos de ouro de diâmetro distintos, cada qual com um furo no meio. Os discos estavam empilhados em uma das torres de modo que acima de cada disco estivesse apenas um disco de diâmetro menor. Os monges receberam então a incumbência de transferir a pilha de discos para uma segunda torre, com o auxílio da terceira torre, seguindo apenas duas regras: os discos deveriam ser movimentados individualmente, e um disco maior nunca deveria ficar sobre um disco menor. O fim da tarefa marcaria então o fim do mundo.

Vamos agora tentar modelar o problema de forma recursiva, para no próximo capítulo resolvê-lo.

Comecemos estudando o problema para uma quantidade de discos igual a 4. Para um total de n discos denotaremos a quantidade de movimentos por $Q(n)$. Assim:

Se $n = 1$, então $Q(1) = 1$. Ou seja, a quantidade de movimentos para levar o único disco do pino A para o pino C é de apenas um movimento.

Se $n = 2$, onde 1 é o disco menor e 2 o disco maior, então os passos são: levar o disco 1 para o pino B ($1B$), levar o disco 2 para o pino C ($2C$) e colocar o disco 1 sobre o 2 em C ($1C$), então $Q(2) = 3$.

Para $n = 3$, sendo os discos 1, 2 e 3, ordenados de forma crescente, temos que a ordem e a menor quantidade de movimentos para colocar os discos no pino C são: $1C$, $2B$, $1B$, $3C$, $1A$, $2C$ e $1C$, sendo $Q(3) = 7$;

Para $n = 4$, onde a ordem crescente é 1, 2, 3 e 4. Os movimentos são: $1B$, $2C$, $1C$, $3B$, $1A$, $2B$, $1B$, $4C$, $1C$, $2A$, $1A$, $3C$, $1B$, $2C$ e $1C$, assim $Q(4) = 15$.

Assim, temos como resposta do problema: são necessários 15 movimentos para levar os quatro discos da haste A para a C.

Vamos desconstruir a solução acima. Para $n = 4$ observe que os primeiros passos ($1B$, $2C$, $1C$, $3B$, $1A$, $2B$, $1B$) são os mesmos utilizados no caso $n = 3$ mudando apenas

as torres utilizadas. Estes passos transferem os três discos menores da torre A para a B. A seguir movemos o disco 4 para a torre C. E terminamos fazendo a sequência ($1C$, $2A$, $1A$, $3C$, $1B$, $2C$ e $1C$) que repete o procedimento de $n = 3$ para levar os pinos de B para C.

Esta mesma análise é válida para $n = 3$ e para $n = 2$.

Deste modo podemos generalizar o problema e tratar o caso geral para n discos.

Neste caso, para n discos, primeiro movemos os $(n-1)$ discos menores para o pino B, depois colocamos o disco n no pino C, e teminamos colocando os $(n-1)$ discos do pino B sobre o disco n no pino C. Assim temos

$$Q(n) = Q(n-1) + 1 + Q(n-1)$$

e

$$\begin{cases} Q(n) = 2Q(n-1) + 1, & n \geq 2. \\ Q(1) = 1. \end{cases}$$

Cálculos que apresentaremos no capítulo 2 nos permitem concluir que

$$Q(n) = 2^n - 1, \quad n \geq 1 \quad \text{(fórmula explicita ou solução fechada).}$$

Construindo uma tabela para alguns valores de n obtemos

n	1	2	3	4	5	$\cdots$
$Q(n)$	$2.0 + 1 = 1$	$2.1 + 1 = 3$	$2.3 + 1 = 7$	$2.7 + 1 = 15$	$2.15 + 1 = 31$	$\cdots$
$Q(n)$	$2^1 - 1 = 1$	$2^2 - 1 = 3$	$2^3 - 1 = 7$	$2^4 - 1 = 15$	$2^5 - 1 = 31$	$\cdots$

No caso dos monges, onde $n = 64$, a quantidade de movimentos é $Q(64) = 2^{64} - 1 =$ 18.446.744.073.709.551.615. Supondo que cada movimento leve 1 microssegundo (significa efetuar 1 000 000 de movimentos em um segundo) , então o tempo necessário para efetuar todos os movimentos é de um pouco mais de 213.503.982 dias que é um número maior que 584.942 anos.

No próximo capítulo iremos verificar e provar as equações explícitas apresentadas aqui, utilizando algumas técnicas conhecidas pelos alunos e outras que serão apresentadas com essa finalidade.

1.3.2 *O problema de Josephus*

Segundo Jane Bichmacher de Glasman em [7], Yossef ben Matitiahu Ha-Cohen, um dos líderes da 1^a Revolta Judaica contra o Império Romano termina seus dias em Roma, onde adota o nome de Flavius Josephus. Lá escreve a história e apologia da Nação Judaica e de si mesmo, suspeito tanto aos olhos de seus correligionários, quanto aos olhos dos romanos.

Em 20 de julho de 67 EC (Era Comum, o mesmo que depois de Cristo, DC), dia da queda de Jotapata (aldeia judaica), ocorre o mais suspeito episódio da vida de Josephus, descrito por ele na terceira pessoa.

> "...Depois da queda da cidade, fugindo em meio aos inimigos, ele [Josephus] desceu a um poço muito profundo ao lado do qual havia uma caverna espaçosa, que não podia ser vista do alto. Lá encontrou 40 dos mais valentes dos seus, que também ali se tinham refugiado e que tinham todo o necessário para vários dias(...). Dois dias assim se passaram; no terceiro, uma mulher o denunciou(...). Vespasiano mandou Paulino e Galicano, dois tribunos, garantir-lhe que o trataria bem, exortando-o a sair; ele não quis fazê-lo, porque, não estando persuadido da clemência dos romanos, e sabendo do seu ressentimento pelo mal que lhes havia feito, temia que quando o tivessem em seu poder, procurassem vingar-se." [7]

Não conseguindo convencê-lo a se entregar, os romanos decidiram incendiar a caverna, só não o fazendo porque Vespasiano o queria vivo, e convida Josephus a se render, prometendo poupá-lo. Perante seus companheiros, aceitar tal proposta seria uma traição: todos prefeririam morrer a se entregar. Josephus dissuadiu-os do suicídio e propôs que se estrangulassem reciprocamente numa ordem determinada por sorteio.

> "Para ver quem deverá ser morto por primeiro por aquele que o seguirá; continuemos a fazer sempre do mesmo modo, a fim de que nenhum de nós se mate por si mesmo, mas receba a morte das mãos de outro." [7]

Restaram vivos somente ele e um companheiro, como ele mesmo tenta explicar, constrangido:

> "Isso continuou até que restavam somente Josephus e outro; o que aconteceu, talvez, por uma especial proteção de Deus ou por casualidade." [7]

Teria havido um truque ao tirar a sorte? Feito prisioneiro, Josephus prediz a Vespasiano que ele ostentaria em breve a púrpura imperial; quando tal se confirmou, em 69 EC, foi liberto, como recompensa, e passou a ser protegido dos romanos.

SOLUÇÃO: O problema inicial é uma das variantes encontradas na literatura da situação apresenta pelo próprio Josephus segundo a escritora. Vamos apresentar a resolução do problema para o número sorteado 2 e considerando o início da contagem para exclusão pelo soldado 1 (primeiro soldado).

Seja n o número de soldados, onde cada número natural representa um soldado e sua posição no círculo inicial e vamos representar por $J(n)$ o soldado vitorioso (aquele que fica por último).

Assim: Para $n = 1$, existe somente o soldado 1 , logo $J(1) = 1$;

Para $n = 2$, existem os soldados 1 e 2, sendo que o primeiro elimina o soldado 2, então $J(2) = 1$.

Verificando-se para $n = 2^m$ a solução é $J(2^m) = 1$.

Demonstração:

Para $m = 0$ ou 1 já estão verificados, então para $m = 2, 3, 4, 5, \ldots$, pode-se verificar que após a primeira rodada de eliminação os soldados "pares" são eliminados (sempre) e a quantidade de soldados restantes é $n = 2^{m-1}$ que é par e fazendo a renumeração dos soldados somente o soldado 1 não é renumerado. Com uma nova rodada de eliminação, restam $n = 2^{m-2}$ soldados, sendo n par e, de maneira análoga, renomeando-se os soldados restantes a partir do segundo soldado e fazendo as demais rodadas de eliminação continuam restando um número par de soldados do tipo $n = 2^m$, até chegar a $n = 2^0 = 1$ que tem como vencedor o primeiro soldado (1) então,

$$J(2^m) = J(2^{m-1}) = J(2^{m-2}) = \cdots = J(2^1) = J(2^0) = 1.$$

Iniciando a construção de uma tabela de resultados de $J(n)$ para $n = 18$, tem-se:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$J(n)$	1	1		1				1								1		

Para $n = 2^m + 1$, onde $m = 1, 2, 3, 4, \ldots$ (observe que para $m = 0$, o caso já foi estudado) a quantidade de soldados pares é 2^{m-1} e de soldados ímpar é $2^{m-1} + 1$ e fazendo a primeira rodada de eliminação, os pares são eliminados e o último soldado (que é ímpar) elimina o soldado 1, assim restam $(2^{m-1} + 1) - 1 = 2^{m-1}$ soldados. Voltamos assim à situação de $n = 2^{m-1}$, onde o vencedor é o primeiro soldado, mas com a diferença de que desta vez o primeiro soldado é o de número 3, já que o soldado de número 1 havia sido eliminado. Segue assim que e $J(2^m + 1) = 3$.

Assim, fazendo $J(2^m + 1) = 3$ a tabela fica:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$J(n)$	1	1	3	1	3			1	3							1	3	

Para $n = 2^m + 2$, onde $m = 2, 3, 4, 5, \ldots$ (pois, para $m = 0, 1$ já foram verificadas as soluções) o número de soldados é par, assim após a primeira rodada de eliminação restam os soldados "impares", com um total de $2^{m-1} + 1$ soldados. O próximo soldado eliminado é o de número 3. Isso nos deixa mais uma vez com um total de 2^{m-1} soldados, mas desta vez o soldado número 5 é o primeiro da fila. Segue assim que $J(2m + 2) = 5$, e podemos mais uma vez atualizar nossa tabela.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$J(n)$	1	1	3	1	3	5		1	3	5						1	3	5

Para $n = 2^m + 3$, onde $m = 2, 3, 4, \ldots$, seguindo o mesmo raciocínio, após uma primeira rodada de eliminações restam os soldados "impares" em um total de $2^{m-1} + 2$ soldados. A seguir, assim como no caso $n = 2^m + 1$, como o total de soldados era ímpar, o soldado 1 é eliminado pelo último soldado, e seguindo a regra o soldado 3 elimina o número 5. Restando agora 2^{m-1} soldados, sendo o de número 7 o primeiro da sequência. Assim, $J(2^m + 3) = 7$ para $m = 2, 3, 4, \ldots$ e a tabela fica:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$J(n)$	1	1	3	1	3	5	7	1	3	5	7					1	3	5

Analogamente, a construção da tabela com número n de soldados o $J(n)$ em cada caso será:

- Para $n = 2^m + 4$, (com $m = 3, 4, 5, \ldots$) temos $J(2^m + 4) = J(2^m + 3) + 2 = 7 + 2 = 9$;
- Para $n = 2^m + 5$, (com $m = 3, 4, 5, \ldots$) temos $J(2^m + 5) = J(2^m + 4) + 2 = 9 + 2 = 11$;

- Para $n = 2^m + 6$, (com $m = 3, 4, 5, \ldots$) temos $J(2^m + 6) = J(2^m + 5) + 2 = 11 + 2 = 13$;
- Para $n = 2^m + 7$, (com $m = 3, 4, 5, \ldots$) temos $J(2^m + 7) = J(2^m + 6) + 2 = 13 + 2 = 15$;
- $\ldots$

Assim, pode-se construir todos os valores possíveis de n, onde a tabela fica:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$J(n)$	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1	3	5

A seguir, daremos outro argumento para cálculo de $J(n)$ e mostraremos que, escrevendo $n = 2^m + p$, onde m é o maior número natural possível e $p \in \{0, 1, 2, \ldots, 2^{m-1} - 1\}$, o soldado vencedor será $J(n) = 2p + 1$. Ou seja, o soldado que irá montar o cavalo e buscar ajuda será aquele que ocupa a posição $2p + 1$, sendo que a quantidade de soldados é $n = 2^m + p$, onde m é o maior expoente da base 2 menor ou igual a n.

SEGUNDA ANÁLISE Iniciamos com o caso $n = 2^m$ onde, seguindo os mesmos argumentos da análise anterior verificamos que

$$J(2^m) = 1.$$

Voltamos nossa atenção para o caso onde $n = 2k$ é par, mas não da forma $n = 2^m$. Neste caso existem k soldados "pares" e k soldados "ímpares".

Após uma primeira rodada de eliminações os soldados pares são eliminados restando apenas k soldados impares. Isso relaciona a instância $2k$ com a instância k do mesmo problema a não ser por uma pequena diferença: os soldados restantes são todos ímpares, e na representação que estabelecemos usamos enumeração sequencial sem desconsiderar nenhum número.

Para resolver o problema vamos *mudar a etiqueta* de cada soldado restante de modo a recuperar a representação que queremos. Para isso o soldado 1 receberá a etiqueta $1'$, o soldado 3 receberá a etiqueta $2'$, o 5 ficará com a etiqueta $3'$ e assim por diante.

Observe a Figura 7 para entender melhor o processo que acabamos de descrever.

Segue assim que o soldado escolhido na instância $2n$ será o mesmo que da instância n, mas agora com nova *etiqueta*. Ou seja,

$$J(2n) = [J(n)]' = 2J(n) - 1. \tag{1.1}$$

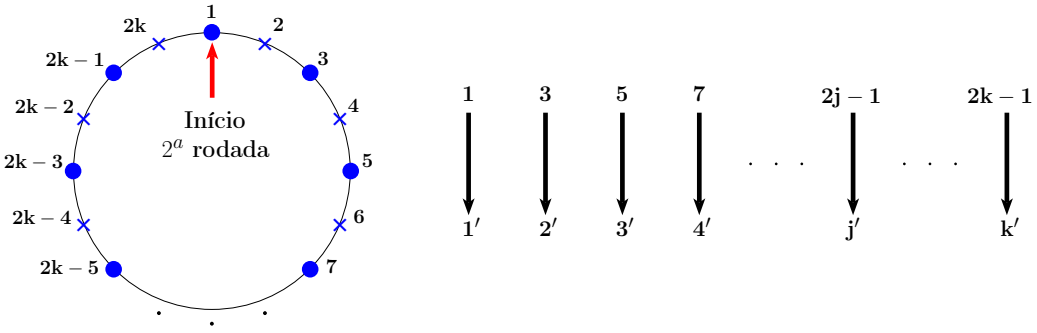


Figura 7: Reenumeração de etiquetas no problema de Josephus - caso n par

Seja n ímpar da forma $n = 2k+1$, então existem k soldados "pares" e $k+1$ soldados "ímpares". Do mesmo modo que antes, a primeira rodada elimina todos os soldados pares, mas desta vez o último soldado é ímpar, fazendo com que o soldado de número 1 seja o próximo eliminado. Desta forma eliminados todos os k soldados pares, mais o soldado número 1, ficando com apenas k soldados ímpares, sendo o soldado 3 o primeiro da sequência. Mais uma vez, para podermos chamar a instância k do problema precisamos reenumerar os soldados, atribuindo a eles uma nova etiqueta, como indicado na Figura 8.

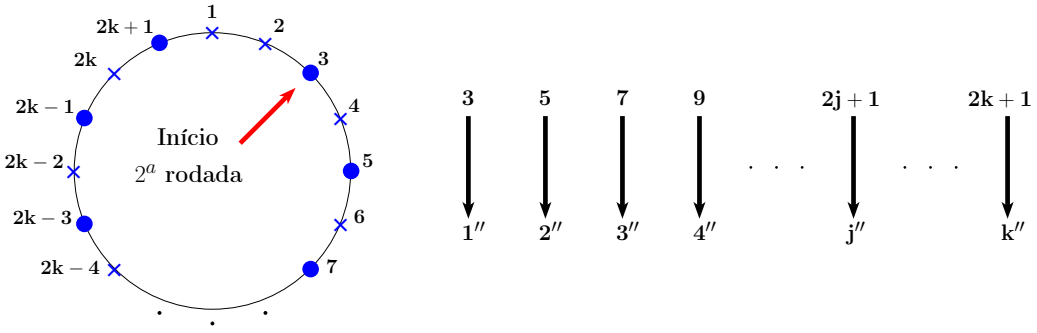


Figura 8: Reenumeração de etiquetas no problema de Josephus - caso $2n+1$ ímpar

Segue assim que

$$J(2n+1) = [J(n)]'' = 2J(n) + 1. \tag{1.2}$$

Tome agora n ímpar, mas da forma $n = 2k-1$. Neste caso existem $k-1$ soldados "pares" e k soldados "ímpares". Assim como no caso anterior, começamos eliminado todos os pares, o que nos deixa com k soldados ímpares. Como queremos relacionar

a instância $2k-1$ com a instância k, faremos a mudança de etiqueta antes de eliminar o soldado 1.

Note agora que precisamos reenumerar os soldados de forma consistente com a representação que adotamos inicialmente. Na nossa representação original, sempre que existem mais de 2 soldados na fila, o segundo soldado é sempre o primeiro a ser eliminado. Nós já eliminamos todos os soldados pares, e dos k soldados restantes o primeiro a ser eliminado será o soldado 1. Com isso, para que a nova enumeração seja consistente, precisamos que o soldado 1 receba a etiqueta $2'''$. A Figura 9 deve esclarecer os argumentos acima.

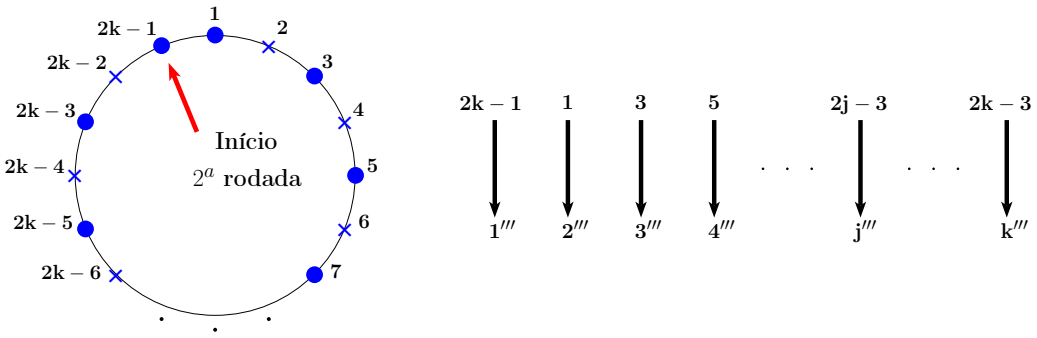


Figura 9: Reenumeração de etiquetas no problema de Josephus - caso $2n-1$

Segue que

$$J(2k-1) = [J(k)]''' = 2J(k) - 3. \tag{1.3}$$

Das equações (1.1) e (1.3) temos que

$$J(2k) = 2J(k) - 1 \quad \text{e} \quad J(2k-1) = 2J(k) - 3,$$

e portanto

$$J(2k) = 2J(k) - 1 - 2 + 2 = [2J(k) - 3] + 2 = J(2k-1) + 2$$

Agora, se $n = 2k$ isto é o mesmo que

$$J(n) = J(n-1) + 2 \tag{1.4}$$

Do mesmo modo, as equações (1.2) e (1.1) dizem que

$$J(2k+1) = 2J(k) + 1 \quad \text{e} \quad J(2k) = 2J(k) - 1,$$

de onde segue que

$$J(2k+1) = 2J(k) - 1 + 1 + 1 = [2J(k) - 1] + 2 = J(2k) + 2.$$

Assim, se $n = 2k + 1$ temos

$$J(n) = J(n-1) + 2,$$

que é igual a (1.4).

Resumidamente, temos

$$J(n) = \begin{cases} 1 & , \quad \text{se } n = 2^m \text{ para algum } m \geq 0; \\ J(n-1) + 2 & , \quad \text{se } n \geq 3 \text{ e } n \neq 2^m, \forall m \geq 0. \end{cases} \tag{1.5}$$

1.3.3 *O problema dos coelhos*

O título Liber Abaci que significa o "Livro do Cálculo", é um livro histórico sobre aritmética escrito por Leonardo Fibonacci (ou Leonardo de Pisa) em 1202. A segunda edição, de 1228, é a que hoje é conhecida.

A teoria contida em Liber Abaci é ilustrada com muitos problemas que representam uma grande parte do livro. Nele aparece um problema que envolve o crescimento de uma população hipotética de coelhos com base em pressupostos idealizados. A solução, de geração em geração, era uma sequência de números que mais tarde conhecida como sequência de Fibonacci. A sequência numérica era conhecida por matemáticos indianos já no século VI, mas foi o Liber Abaci que a introduziu no Ocidente. (Ver [8])

SOLUÇÃO: Ao fixar como mês um o início do processo, tem-se, no início do primeiro mês, um único casal jovem. Já no segundo mês, esse casal será adulto. Considerando-se que um par adulto produz um novo par a cada mês, no início do terceiro mês existirão dois pares de coelhos, sendo um par adulto e outro recém-nascido.

No início do quarto mês o par adulto produzirá mais um par, enquanto que o outro par completará um mês de vida e ainda não estará apto a reproduzir. Assim,

existirão três pares de coelhos, sendo um par adulto, um par com um mês de idade e mais um par recém-nascido.

No início do quinto mês existirão dois pares adultos, sendo que cada um deles já reproduz um novo par e mais um par que completou um mês de vida. Logo, existirão cinco pares.

No início do sexto mês existirão três pares adultos, sendo que cada um reproduz um novo par e mais dois pares que completam um mês de vida. Logo, existirão oito pares.

Seguindo-se o mesmo raciocínio para os outros meses, obtém-se a famosa Sequência de Fibonacci, cujos primeiros termos são: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, . . ., tal sequência recebe o nome "Fibonacci" devido ao apelido dado a Leonardo de Pisa por Baldassare Boncompagni, seu editor de trabalhos no século XIX, que signfica "filho de Bonaccio".

Fibonacci tornou-se conhecido devido a este problema que existia no seu livro "Liber Abaci". A solução deste problema é uma sequência numérica e um matemático francês, Edouard Lucas, ao editar um trabalho seu, ligou o nome de Fibonacci a essa sequência. [9]

Uma análise rápida mostra que cada termo da sequência acima é dado recursivamente pela expressão:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2),\ n \geq 2 \end{aligned}$$

onde n é a quantidade de meses.

No nosso problema queremos saber quantos pares de coelhos um casal pode ter em 12 meses, assim temos:

$$F(12) = F(12-1) + F(12-2)$$
$$F(12) = F(11) + F(10)$$
$$F(11) = 89$$
$$F(10) = 55$$
$$F(12) = 89 + 55$$
$$F(12) = 144.$$

Os valores para $F(10)$ e $F(11)$ são calculados da mesma forma, a partir dos valores de $F(9)$ e $F(8)$, que por sua vez são calculados a partir dos valores anteriores, seguindo assim até chegar a $F(0)$ e $F(1)$ que são conhecidos.

Tal raciocínio fica melhor explicitado na tabela abaixo.

n	0	1	2	3	4	5	6	7	8	9	10	11	12
$F(n)$	0	1	1	2	3	5	8	13	21	34	55	89	144

Temos assim nossa resposta: Partindo de um casal de coelhos serão gerados em doze meses um total de 143 casais de coelhos.

A sequência de Fibonacci tem uma solução explicita dada por

$$F(n) = \frac{\sqrt{5}}{5}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right],$$

que será verficada no próximo capítulo.

Para encerrar o capítulo, vamos deixar alguns exercícios propostos aos leitores para a verificação de outras relações de recorrência com a finalidade de enriquecer o assunto tratado aqui.

1.4 ATIVIDADES

1. A visita que M. C. Escher fez à Alhambra (Espanha), em 1922, inspirou os seus trabalhos seguintes. Conheça Alhambra visitando a página `http://pt.wikipedia.org/wiki/Alhambra` e em seguida, a página oficial de M.C.Escher em `http://www.mcescher.com` para verificar e apreciar o trabalho do mesmo.

2. Faça uma pesquisa na internet sobre Fractal e veja alguns fractais feitos com o uso do computador.

3. O lixo em um depósito sanitário decompõe-se a uma taxa de 5% ao ano. Se 100 toneladas de lixo são inicialmente depositadas no depósito, quanto permanece não decomposto após 20 anos (ou seja, no início do 21^o ano)? (Dica: Enuncie e resolva uma relação de recorrência.)

4. Membros antigos da Sociedade de Pitágoras definiram números figurados como sendo o número de pontos em certa configuração geométrica. Os primeiros números triangulares são 1, 3, 6, e 10, e são semelhantes ao diagrama da figura abaixo:

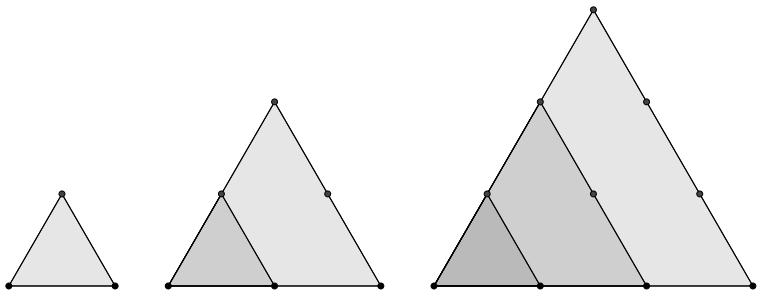


Figura 10: Números Triangulares

Observe que cada número da sequência refere-se à quantidade de pontos que são os vértices dos triângulos existentes.

Construa geometricamente mais alguns números triangulares e encontre a fórmula para o n-ésimo número triangular.

5. Os primeiros números quadráticos são 1, 4, 9, 16, e são definidos pelas figuras abaixo.

Mostre que o n-ésimo número quadrático é dado por n^2.

6. Os primeiros números pentagonais (veja a figura anterior) são $1, 5, 12, 22$, e são semelhantes ao diagrama da figura abaixo:

Encontre a fórmula para o n-ésimo número pentagonal.

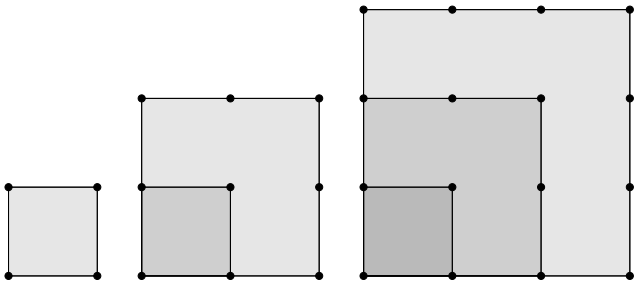


Figura 11: Números Quadráticos

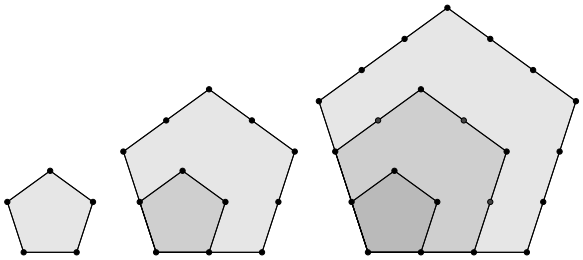


Figura 12: Números Pentagonais

7. De quantas maneiras podemos formar palavras com n letras utilizando o alfabeto $\{a, b, c, d\}$ se as letras a e b não devem aparecer juntas?

8. (Itália/1996) Dado o alfabeto com três letras a, b, c, encontre o número de palavras com n letras contendo um número par de a's. Considere $n \geq 2$.

9. Conheça e resolva algumas das variantes do problema de Josephus apresentados no site: `http://www.calendario.cnt.br/Paginas/Josefo.htm`

2

CONCEITOS TEÓRICOS

"Ao tentar resolver o problema, encontrei obstáculos dentro de obstáculos. Por isso, adotei uma solução recursiva."
— Um aluno

2.1 RELAÇÕES DE RECORRÊNCIA

Uma relação de recorrência, ou equação de recorrência é uma equação que define uma sequência recursivamente. Mais formalmente, vale a seguinte definição.

Definição 2.1. Uma relação de recorrência de ordem k é uma equação do tipo

$$x_n = f_1 \cdot x_{n-1} + f_2 \cdot x_{n-2} + \cdots + f_k \cdot x_{n-k} + g(n), \qquad n \geq k+1,$$

onde

$$x_1, x_2, \ldots, x_k,$$

são valores dados, também chamados de condições iniciais, e

$$f_i = f_i(i, x_{n-1}, \ldots, x_{n-k}), \qquad i = 1, \ldots, k,$$

são funções também fixadas.

Os exemplos mais conhecidos de relações de recorrência de ordem 1 são as progressões aritmética e geométrica. A equação de Fibonacci apresentada no capítulo anterior, onde

$$x_n = x_{n-1} + x_{n-2},$$

é um exemplo de equação de segunda ordem.

Uma *solução* de uma equação de recorrência é uma função que expressa x_n como função do índice n. Ou seja:

Definição 2.2. Uma solução da equação de recorrência

$$x_n = f_1 \cdot x_{n-1} + f_2 \cdot x_{n-2} + \cdots + f_k \cdot x_{n-k} + g(n), \qquad n \geq k+1,$$

é uma função $F : \mathbb{N} \to \mathbb{R}$ tal que

$$x_n = F(n), \quad n \geq 1.$$

Neste capítulo introduziremos alguns tipos especiais de equações de recorrência, e estudaremos algumas técnicas para encontrar sua solução. Para começar vamos tratar das já conhecidas P.A. e P.G..

Exemplo 2.1. Solução de uma P.A.

Para encontrar a solução de uma P.A. do tipo

$$a_n = a_{n-1} + r,$$

podemos proceder da seguinte maneira.

$$\begin{array}{rcccccccl}
a_2 &=& a_1 + r & & & & & & \\
a_3 &=& a_2 + r &=& a_1 + r + r &=& a_1 + 2r & & \\
a_4 &=& a_3 + r &=& a_1 + 2r + r &=& a_1 + 3r & & \\
& & & & \vdots & & & & \\
a_n &=& a_{n-1} + r &=& a_{n-2} + 2r &=& \cdots &=& a_1 + (n-1)r.
\end{array}$$

E desta forma, a solução de uma P.A. de razão r e primeiro termo a_1 é dado por

$$a_n = a_1 + (n-1)r; \qquad n \geq 1.$$

Outro modo de encontrar a solução de uma P.A. é através da chamada *soma telescópica*, mas este assunto será tratado um pouco mais a frente.

Exemplo 2.2. Solução de uma P.G. Considere uma P.G. de razão q e valor inicial x_1 dada por

$$x_n = q \cdot x_{n-1}.$$

Procedendo de maneira análoga a uma P.A., encontramos que

$$\begin{array}{rcrcrcrcl}
x_2 & = & x_1 \cdot q & & & & & & \\
x_3 & = & x_2 \cdot q & = & x_1 \cdot q \cdot q & = & x_1 \cdot q^2 & & \\
x_4 & = & x_3 \cdot q & = & x_1 \cdot q^2 \cdot q & = & x_1 + \cdot q^3 & & \\
 & & & & \vdots & & & & \\
x_n & = & x_{n-1} \cdot q & = & x_{n-2} \cdot q^2 & = & \cdots & = & x_1 \cdot q^{n-1}.
\end{array}$$

e portanto

$$x_n = x_1 \cdot q^{n-1}.$$

2.1.1 *Classificação das Equações de Recorrência*

Relações de recorrência são objetos complexos, cuja solução explícita pode requerer técnicas complicadas e sofisticadas de cálculo. Existem, inclusive, diversas equações cuja solução explícita não sabemos calcular. Podemos, no entanto, classificar equações de recorrência em classes, cujas soluções podem ser encontradas por técnicas similares.

Definição 2.3. Considere uma relação (equação) de recorrência de ordem k dada por

$$x_n = f_1 \cdot x_{n-1} + f_2 \cdot x_{n-2} + \cdots + f_k \cdot x_{n-k} + g(n), \qquad n \geq k+1.$$

- Quando $g(n) = 0$, para todo n, dizemos que a equação é homogênea;
- Quando $g(n) \neq 0$, para algum n, dizemos que a equação não é homogênea;
- Quando $f_i = f(i), 0 < i \leq n$, diremos que a equação é linear;

- Quando f_i é constante, diremos que a equação é linear com coeficientes constantes;
- Quando $f_i = f_i(i, x_{n-1}, \ldots, x_{n-k})$, para algum $i = 1, \ldots, k$, trata-se de uma equação não linear;

Ao longo do primeiro capítulo vimos alguns exemplos de equações de recorrência lineares. Neste trabalho trataremos apenas deste tipo de equações, nos concentrando principalmente em equações lineares de primeira ordem.

Seguem abaixo alguns exemplos para fixação das definições acima

Exemplo 2.3. Classificação de algumas equações de recorrência

- A equação de Fibonacci
$$x_{n+2} = x_{n+1} + x_n$$
é uma equação linear de ordem 2 homogênea de coeficientes constantes.
- $F(n) = rF(n-1)\,(1 - F(n-1))$, conhecida como *equação logística* é uma equação não linear homogênea.
- $x_n = 2x_{n-1} + 3$ é uma equação linear de ordem 1, não homogênea de coeficientes constantes.
- $x_n = nx_{n-1} + n^2x_{n-2}$ é uma equação linear não homogênea.

2.2 EQUAÇÕES LINEARES DE 1^a ORDEM

De acordo com a definição 2.3 uma relação de recorrência de primeira ordem é uma equação do tipo
$$x_n = f(n)x_{n-1} + g(n).$$

Os exemplos mais comuns, e constantes do currículo atual do ensino médio, são as progressões aritmética e geométrica (P.A. e P.G.), já mencionadas diversas vezes

neste texto. Para encontrar uma P.A. fazemos $f(n) = 1$ e $g(n) = r$ (a razão da P.A.) para todo n. No caso de uma P.G. invertemos esta lógica, fazendo $f(n) = q$ (a razão da P.G.) e $g(n) = 1$ para todo n.

As soluções para estes dois tipos de equação foram dadas no início deste capítulo, e já são bem conhecidas por estudantes de ensino médio.

A seguir enumeraremos alguns tipos especiais de equações de primeira ordem, e algumas técnicas usadas para encontrar sua solução explícita.

2.2.1 *Somas Telescópicas*

Considere uma equação linear de primeira ordem da forma

$$x_{n+1} = x_n + g(n). \tag{2.1}$$

Escrevendo esta equação para os vários índices de 1 até n obtemos

$$\begin{aligned} x_{n+1} &= x_n + g(n) \\ x_n &= x_{n-1} + g(n-1) \\ x_{n-1} &= x_{n-2} + g(n-2) \\ \vdots &\quad \vdots \\ x_3 &= x_2 + g(2) \\ x_2 &= x_1 + g(1). \end{aligned}$$

Somando todas estas expressões encontramos

$$x_{n+1} + (x_n + \cdots + x_2) = (x_n + \cdots + x_2) + x_1 + g(1) + g(2) + \cdots + g(n-1),$$

que pode ser simplificada até

$$x_{n+1} = x_1 + \sum_{k=1}^{n} g(k).$$

Encontramos assim que a solução da equação (2.1) é dada por

$$x_n = x_1 + \sum_{k=1}^{n-1} g(k). \tag{2.2}$$

Exemplo 2.4. P.A. como Soma Telescópica

Observe que uma P.A. de razão r é uma equação to tipo de (2.1), onde $g(n) = r$ para todo n. Deste modo temos que

$$x_n = x_1 + \sum_{k=1}^{n-1} r = x_1 + (n-1)r.$$

Esta técnica, onde somamos equações simplificando termos similares, é conhecida como soma telescópica, e é bastante utilizada na solução de diversas equações de recorrência, como veremos a seguir.

2.2.2 *Equações de 1^a Ordem com Coeficiente Constante*

Vamos agora considerar equações do tipo

$$x_{n+1} = q \cdot x_n + g(n), \tag{2.3}$$

onde $q \in \mathbb{R}$ é uma constante.

Para resolver esta equação, primeiro reescreva (2.3) como

$$x_{n+1} - q \cdot x_n = g(n),$$

e liste todas as equações para indices de 1 a n, obtendo

$$\begin{aligned} x_{n+1} - q \cdot x_n &= g(n) \\ x_n - q \cdot x_{n-1} &= g(n-1) \\ x_{n-1} - q \cdot x_{n-2} &= g(n-2) \\ \vdots \quad &\quad \vdots \\ x_3 - q \cdot x_2 &= g(2) \\ x_2 - q \cdot x_1 &= g(1). \end{aligned}$$

Observe que neste momento não podemos nos aproveitar da soma telescópica, pois todo termo x_i aparece na equação anterior como $q \cdot x_i$, impedindo o cancela-

mento. Para resolver este problema, multiplique cada equação acima por $1, q, q^2, \ldots, q^{n-1}$, respectivamente, obtendo

$$\begin{aligned}
x_{n+1} - q \cdot x_n &= g(n) \\
q \cdot x_n - q^2 \cdot x_{n-1} &= q \cdot g(n-1) \\
q^2 \cdot x_{n-1} - q^3 \cdot x_{n-2} &= q^2 \cdot g(n-2) \\
&\vdots \\
q^{n-2} \cdot x_3 - q^{n-1} \cdot x_2 &= q^{n-2} \cdot g(2) \\
q^{n-1} \cdot x_2 - q^n \cdot x_1 &= q^{n-1} \cdot g(1).
\end{aligned}$$

e somando obtemos

$$\begin{aligned}
x_{n+1} - \cancel{q \cdot x_n} &= g(n) \\
\cancel{q \cdot x_n} - \cancel{q^2 \cdot x_{n-1}} &= q \cdot g(n-1) \\
\cancel{q^2 \cdot x_{n-1}} - \cancel{q^3 \cdot x_{n-2}} &= q^2 \cdot g(n-2) \\
&\vdots \\
\cancel{q^{n-2} \cdot x_3} - \cancel{q^{n-1} \cdot x_2} &= q^{n-2} \cdot g(2) \\
\cancel{q^{n-1} \cdot x_2} - q^n \cdot x_1 &= q^{n-1} \cdot g(1) \\
\hline
x_{n+1} - q^n x_1 &= g(n) + q \cdot g(n-1) + \cdots + q^{n-1} \cdot g(1).
\end{aligned}$$

Ou seja,

$$x_{n+1} = q^n x_1 + \sum_{k=1}^{n} q^{n-k} \cdot g(k), \tag{2.4}$$

ou ainda

$$x_{n+1} = q^n x_1 + \sum_{k=0}^{n-1} q^k \cdot g(n-k). \tag{2.5}$$

Exemplo 2.5. Considere a equação

$$\begin{cases} x_{n+1} = 3x_n + 1 \quad \text{para } n \geq 1 \\ x_1 = 2 \end{cases}.$$

Neste caso temos $g(n) = 1,\ x_1 = 2$ e $q = 3$ o que nos leva a

$$\begin{aligned} x_{n+1} &= 3^n \cdot 2 + \sum_{k=0}^{n-1} 3^k \cdot 1 \\ &= 2 \cdot 3^n + (1 + 3^2 + \cdots + 3^{n-1}) \\ &= 2 \cdot 3^n + \frac{3^n - 1}{2} \\ &= \frac{5}{2} \cdot 3^n - \frac{1}{2} \end{aligned}$$

e portanto

$$x_n = \frac{5}{2}\, 3^{n-1} - \frac{1}{2}, \quad n \geq 1$$

Uma classe interessante de equações de primeira ordem com coeficientes constantes são as chamadas *Progressões Aritmético-Geométricas* (P.A.G.), dadas por

$$x_{n+1} = q \cdot x_n + r \cdot q^n, \quad n \geq 1. \tag{2.6}$$

Neste caso, temos uma equação linear de primeira ordem e coeficiente constante, onde $g(n) = r \cdot q^n$, e portanto a solução é dada por

$$\begin{aligned} x_{n+1} &= q^n \cdot x_1 + \sum_{k=1}^{n} q^{n-k} \cdot r \cdot q^k \\ &= q^n \cdot x_1 + r \sum_{k=1}^{n} q^n \\ &= q^n \cdot x_1 + r\ n\ q^n \\ &= q^n \left[x_1 + n\, r\right] \end{aligned}$$

Temos assim que a solução de (2.6) é dada por

$$x_n = q^{n-1} \left[x_1 + (n-1)\, r\right].$$

Tendo em mãos a solução geral é fácil agora perceber que uma progressão aritmético-geométrica recebe este nome pois é construída multiplicando os termos de uma P.A.

$$(x_1, x_1 + r, x_1 + 2r, \ldots, x_1 + (n-1)r, \ldots)$$

pelos termos respectivos de uma P.G.

$$(1, q, q^2, \ldots, q^{n-1}, \ldots),$$

gerando a sequência

$$(x_n) = (x_1, q[x_1 + r], q^2[x_1 + 2r], \ldots, q^{n-1}[x_1 + (n-1)r], \ldots).$$

Mais sobre P.A.G. pode ser encontrado em [10]

Exemplo 2.6. Considere a P.A.G. dada por

$$\begin{cases} a_{n+1} = 2\,a_n + 3 \cdot 2^n \quad , n \geq 1 \\ a_1 = 1 \end{cases}$$

Neste caso a solução geral é dada por

$$a_n = 2^{n-1}\left[1 + 3(n-1)\right],$$

e a sequência (a_n) é dada por $(1, 8, 28, 80, \cdots)$.

Vamos agora partir para a solução dos problemas apresentados no Capítulo 1. Lá descrevemos o problema e encontramos as equações de recorrências que modelam a solução. A seguir vamos apresentar a solução final, fechada para o problema.

2.2.3 *Solução do Problema das Torres de Hanói*

Este problema foi descrito na Seção 1.1, e na seção 1.3.1 encontramos a relação de recorrência que o modela. Lembrando, se $Q(n)$ representa o mínimo de movimentos necessários para se mover os pinos de uma torre para a outra, então

$$\begin{cases} Q(n) = 2Q(n-1) + 1, \quad n \geq 2 \\ Q(1) = 1 \end{cases}$$

Temos assim que o problema é modelado por uma relação de recorrência de ordem 1, não-homogênea e de coeficiente constante. Poderíamos agora simplesmente evocar

as equações (2.4) ou (2.5) e apresentar o resultado final. Mas para fixar melhor a técnica, vamos repetir a solução apresentada no caso específico desta equação.

Lembrando, queremos usar a soma telescópica, mas o fator 2 multiplicando o termo do lado direito da equação não permite. O que fazemos então é multiplicar as equações do tipo $Q(n-k) = 2Q(n-k-1)+1$ por 2^k de modo a corrigir os termos e possibilitar a soma telescópica. Temos então que

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

Somando agora todas as equações obtemos

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

Deste modo concluímos que o número de movimentos necessários para mover n pinos entre duas torres no jogo das Torres de Hanói usando a estratégia descrita é de

$$Q(n) = 2^n - 1.$$

2.3 SOLUÇÃO DO PROBLEMA DE JOSEPHUS

Do mesmo modo que o problema das torres, este problema foi apresentado na seção 1.1 e sua relação de recorrência encontrada em 1.3.2. Resumidamente, temos

$$J(n) = \begin{cases} 1 & , \quad \text{se } n = 2^m \text{ para algum } m \geq 0; \\ J(n-1)+2 & , \quad \text{se } n \geq 3 \text{ e } n \neq 2^m, \forall m \geq 0. \end{cases} \tag{2.7}$$

Temos então que a sequência $J(1), J(2), \ldots$ segue uma espécie de progressão aritmética de razão 2 e primeiro termo 1, mas que se reinicia sempre que chega a um termo cujo índice é potência de 2.

Para entender isso melhor fixe $m \geq 0$ e considere a sequência de todos os valores naturais entre 2^m e $2^{m+1} - 1$, dada por

$$2^m, 2^m + 1, 2^m + 2, \ldots, 2^{m+1} - 2, 2^{m+1} - 1.$$

Todo número n neste conjunto pode ser representado por $n = 2^m + p$, com $p \in \{1, 2, \ldots, 2^m - 1\}$ (por que?), e m é portanto **o expoente da maior potência de** 2 **menor que** n.

Reescrevendo a equação (2.7) encontramos

$$J(2^m + p) = \begin{cases} 1 & , \quad \text{se } p = 0; \\ J(2^m + p - 1) + 2 & , \quad \text{se } 0 < p < 2^m. \end{cases} \tag{2.8}$$

Para simplificar, faça $F(p) = J(2^m + p)$ temos

$$F(p) = \begin{cases} 1 & , \quad \text{se } p = 0; \\ F(p - 1) + 2 & , \quad \text{se } 0 < p < 2^m. \end{cases} \tag{2.9}$$

Assim, para $0 \leq p < 2^{m+1}$ temos

$$F(p) = 2p + 1,$$

e portanto

$$J(2^m + p) = 2p + 1, \tag{2.10}$$

para todo $m \geq 0$ e $0 \leq p \leq 2^m - 1$.

Se quisermos escrever (2.10) em função apenas de n, precisamos primeiro determinar m e p em função de n.

Comecemos determinando m. Observe primeiro que pela própria definição de m e p temos

$$0 \leq p < 2^m,$$

e portanto

$$2^m \leq n = 2^m + p < 2^{m+1}.$$

Segue que

$$\log_2(2^m) \leq \log_2 n < \log_2(2^{m+1}),$$

e

$$m \leq \log_2 n < m + 1.$$

Concluímos então que $log_2 n$ é um número real entre m e $m+1$, sendo estritamente menor que $m+1$. Portanto, se tomarmos apenas a parte inteira de $\log_2 n$ encontramos justamente m. Ou seja,

$$m = \lfloor \log_2 n \rfloor,$$

onde $\lfloor x \rfloor$ denota a parte inteira de x.

Temos assim que $n = 2^{\lfloor \log_2 n \rfloor} + p$, o que implica que

$$p = n - 2^{\lfloor \log_2 n \rfloor}.$$

Juntanto tudo temos que

$$J(n) = 2 \left[n - 2^{\lfloor \log_2 n \rfloor} \right] + 1, \tag{2.11}$$

para todo $n \geq 1$.

2.4 EQUAÇÕES LINEARES DE 2^a ORDEM

Considere a equação de segunda ordem dada por

$$x_{n+1} = a\, x_n + b\, x_{n-1}.$$

Defina o vetor coluna

$$v_n = \begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix},$$

e observe que

$$\begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix} = \begin{pmatrix} a\, x_n + b\, x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_n \\ x_{n-1} \end{pmatrix}.$$

Ou seja, se

$$A = \begin{pmatrix} a & b \\ 1 & 0 \end{pmatrix},$$

então

$$v_n = A\, v_{n-1}, \quad \text{com } v_0 = \begin{pmatrix} x_1 \\ x_0 \end{pmatrix}.$$

Segue assim que v_n é uma espécie de P.G. matricial, e seguindo a mesma técnica que usamos para encontrar a solução de uma P.G., temos

$$v_n = A^n v_0.$$

Infelizmente, calcular potência de matrizes pode se tornar uma tarefa árdua, cujas técnicas vão além do escopo deste trabalho. Mas apesar disso, esta pequena descoberta pode servir de fonte de inspiração. Se podemos escrever uma equação linear de ordem 2 não-homogênea e com coeficientes constantes como uma P.G. matricial, é razoável esperar que as soluções da equação sejam similares as soluções de uma P.G..

Com isso em mente, vamos então buscar uma solução do tipo $x_n = c\,q^n$. Substituindo em $x_{n+1} = a\,x_n + b\,x_{n-1}$ obtemos

$$c\,q^{n+1} = a\,c\,q^n + b\,c\,q^{n-1}.$$

e dividindo por $c\,q^{n-1}$ obtemos

$$q^2 - a\,q - b = 0.$$

O polinômio em q dado por $p(q) = q^2 - a\,q - b$ é chamado de *equação característica* da relação de recorrência, suas raízes determinam soluções possíveis para a equação.

No caso em que o polinômio tem duas raízes, q_1 e q_2 obtemos duas soluções distintas: $x_n = c_1\,q_1^n$ e $x_n = c_2\,q_2^2$. Desta forma, a solução geral pode ser escrita como

$$x_n = c_1\,q_1^n + c_2\,q_2^n,$$

e as constante c_1, c_2 dependem das condições iniciais do problema.

Vejamos um exemplo para deixar as coisas mais claras.

Exemplo 2.7. Vamos resolver a equação

$$\begin{cases} a_n = 5a_{n-1} - 6a_{n-2}, & n \geq 2 \\ a_0 = 0,\ a_1 = 2 \end{cases}.$$

Supondo $a_k = q^k$, com $q \neq 0$ obtemos

$$q^n - 5q^{n-1} + 6q^{n-2} = 0$$

e dividindo por q^{n-2} obtemos

$$q^2 - 5q + 6 = 0,$$

cujas raízes são 2 e 3.

Do problema temos que $a_0 = 0$ e $a_1 = 2$, então

$$\begin{cases} a_0 &= c_1 3^0 + c_2 2^0 = 0 \\ a_1 &= c_1 3^1 + c_2 2^1 = 2 \end{cases}.$$

Ou seja

$$\begin{cases} c_1 + c_2 &= 0 \\ 3c_1 + 2c_2 &= 2 \end{cases},$$

e encontramos $c_1 = 2$ e $c_2 = -2$. Portanto

$$a_n = 2 \cdot 3^n - 2 \cdot 2^n = 2 \cdot 3^n - 2^{n+1}.$$

Considere agora o caso em que o polinômio $p(q) = q^2 - aq - b$ possua apenas uma raíz q_1. Neste caso encontramos apenas soluções do tipo $x_n = c_1 q_1^n$, mas isso não parece ser suficiente. De fato, tendo duas condições iniciais, estas podem não ser compatíveis com a solução encontrada. O que fazer neste caso?

Para resolver este problema tome $F(n)$ uma solução qualquer de $x_n + 1 = ax_n + bx_{n-1}$, e defina

$$y_n = \frac{F(n)}{a^n}.$$

Temos assim que $F(n) = y_n q_1^n$, e como $F(n)$ é uma solução da recorrência, segue que

$$F(n+1)q_1^{n+1} = aF(n)q_1^n + bF(n-1)q_1^{n-1}.$$

Antes de continuar, lembre que, se q_1 é a única raíz de $q^2 - aq - b$, então $a = 2q_1$ (soma das raízes) e $-b = q_1^2$ (produto das raízes). Com isso obtemos

$$y_{n+1}q_1^{n+1} = 2y_n q_1^{n+1} - y_{n-1}q_1^{n+1},$$

e

$$y_{n+1} = 2y_n - y_{n-1}.$$

Rearrumando os termos obtemos

$$y_{n+1} - y_n = y_n - y_{n-1}.$$

Observe que portanto a sequência $(y_n - y_{n-1})_{n\geq 1}$ é constante. Fazendo $y_n - y_{n-1} = r$ concluímos que y_n é uma P.A., e assim

$$y_n = nr + y_0.$$

Concluímos assim que $F(n) = r\, n\, q_1^n + y_0\, q_1^n$, e fazendo $y_0 = c_1$ e $r = c_2$ temos

$$F(n) = c_1\, q_1^n + c_2\, n\, q_1^n,$$

e como $F(n)$ era uma solução qualquer segue que toda solução tem esta forma.

Exemplo 2.8. Queremos resolver a equação: $a_n - 4a_{n-1} + 4a_{n-2} = 0$, com $a_0 = 0$ e $a_1 = 2$.

Temos que a equação característica que associada a esta equação de recorrência é

$$q^2 - 4q + 4 = 0,$$

que possui $q = 2$, como raiz dupla da equação.

Assim, a solução é dada por

$$a_n = c_1 2^n + c_2 . n 2^n.$$

Usando as condições iniciais encontramos

$$\begin{cases} a_0 &= c_1 2^0 + c_2 \cdot 0 \cdot 2^0 = 0 \\ a_1 &= c_1 2^1 + c_2 \cdot 1 \cdot 2^1 = 2 \end{cases}$$

O que nos leva a $c_1 = 0$ e $c_2 = 1$, nos dando a solução

$$a_n = n\, 2^n.$$

2.4.1 *Encontrando a Sequência de Fibonacci*

No problema dos coelhos vimos que a relação de recorrência era dada por

$$f_n = f_{n-1} + f_{n-2}, \quad n \geq 2$$

onde tínhamos como condições iniciais que $f_1 = f_2 = 1$.

Temos assim uma relação de recorrência homogênea linear de segunda ordem, cuja equação característica é $q^2 - q - 1 = 0$. Ou seja,

$$\left(q - \frac{1+\sqrt{5}}{2}\right)\left(q - \frac{1-\sqrt{5}}{2}\right) = 0,$$

cujas raízes são

$$q_1 = \frac{1+\sqrt{5}}{2} \quad \text{e} \quad q_2 = \frac{1-\sqrt{5}}{2}.$$

Assim, se

$$f_n = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

podemos determinar c_1 e c_2 utilizando o seguinte sistema

$$\begin{cases} f_1 = c_1\left(\frac{1+\sqrt{5}}{2}\right) + c_2\left(\frac{1-\sqrt{5}}{2}\right) & = 1 \\ f_2 = c_1\left(\frac{1+\sqrt{5}}{2}\right)^2 + c_2\left(\frac{1-\sqrt{5}}{2}\right)^2 & = 1 \end{cases}$$

o que nos leva a

$$c_1 = \frac{1}{\sqrt{5}} \quad \text{e} \quad c_2 = -\frac{1}{\sqrt{5}}$$

e a solução explícita para a sequência de Fibonacci fica

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

2.5 MUDANÇA DE VARIÁVEIS

Em alguns casos podemos fazer uso de soluções conhecidas para tratar de problemas novos. Considere o exemplo a seguir.

Exemplo 2.9. Tome a seguinte equação de recorrência:

$$T(n) = 1 + T(n-1) + T(n-2), \quad n \geq 2,$$

onde $T(1) = T(2) = 0$.

Trata-se de uma recorrência linear não homogênea de segunda ordem, onde

$$T(1) = 0,\ T(2) = 0,\ T(3) = 1,\ T(4) = 2,\ T(5) = 4,\ T(6) = 7,$$

$$T(7) = 12,\ T(8) = 20,\ T(9) = 33,\ T(10) = 54,\ T(11) = 88.$$

Observando os termos da sequência $T(n)$ - $0, 0, 1, 2, 4, 7, 12, 20, 33, 54, 88, \ldots$ - e os da sequência de Fibonacci - $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$ - verificamos que pode ser possível escrever

$$T(n) = a_n - 1, \quad n \geq 1,$$

onde a_n é o n-ésimo termo da sequência de Fibonacci.

De fato, observe que

$$T(n) = T(n-1) + T(n-2) + 1 \iff T(n) + 1 = (T(n-1) + 1) + (T(n-2) + 1),$$

e fazendo $a_n = T(n) + 1$ temos

$$a_n = a_{n-1} + a_{n-2}, \quad n \geq 2$$

e $a_1 = a_2 = 1$.

Portanto

$$a_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n.$$

Ou seja, o termo geral da sequência $T(n)$ é dado por

$$T(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n - 1.$$

No exemplo, fazendo $a_n = T(n) + 1$ encontramos uma sequência cuja solução já era conhecida, e assim resolvemos uma equação aparentemente nova.

A técnica utilizada no problema anterior é conhecida como *mudança de variáveis* e consiste de alterar os termos de uma sequência de modo a encontrar uma nova sequência cuja solução sabemos encontrar, e a partir desta resolver a equação original.

Alguns problemas recursivos são resolvidos quando divididos em dois ou mais subproblemas de mesmo tamanho. Resolvendo-os recursivamente utilizamos seus resultados para resolver o problema original, estratégia conhecida como "dividir e conquistar".

Exemplos de equações de recorrência que surgem em problemas como este serão dados logo a seguir, mas resolver deste tipo pode depender de técnicas um pouco mais avançadas. No entanto, se nos concentrarmos apenas em uma parte da sequência, o método da mudança de variável pode ser utilizado.

Veja alguns exemplos abaixo.

Exemplo 2.10. Considere a recorrência:

$$F(n) = F\left(\frac{n}{2}\right) + 4$$

Podemos observar que não tem sentido tomar n no conjunto $\{1, 2, 3, 4, 5, \ldots\}$, pois $n/2$ não pertence a esse conjunto quando n é ímpar, nem de tomar n no conjunto $\{2, 4, 6, 8, \ldots\}$ pois $(n/2)/2$ não pertence a esse conjunto quando $n/2$ é ímpar, contudo se nessa recorrência tomamos n pertencente ao conjunto $\{2^1, 2^2, 2^3, 2^4, \ldots\}$ das potências inteiras de 2, então a sequência fica bem definida.

Assim, podemos fazer $n = 2^k$ e reescrever a recorrência da seguinte forma:

$$F(n) = F(2^k) = F(2^{k-1}) + 4, \quad \text{para } k = 1, 2, 3, 4, \ldots$$

Como sabemos, há muitas funções F definidas sobre as potências inteiras de 2 que satisfazem essa recorrência. Para cada número i, há somente uma função F que satisfaz a recorrência e tem valor inicial $F(2^0) = i$. Se $i = 6$, por exemplo, teremos:

n	1	2	4	8	16	32	...
$F(n)$	6	10	14	18	22	26	...

Para determinação da fórmula explicita faça $G(k) = F(2^k)$. Temos assim

$$G(k) = G(k-1) + 4,$$

e $G(k)$ é uma P.A., de onde concluímos que

$$G(k) = 4k + G(0).$$

E como $G(0) = F(2^0) = 6$ temos

$$F(2^k) = 4k + 6.$$

Como $n = 2^k$, temos que $k = \log_2 n$. Ou seja

$$F(n) = 6 + 4\log_2 n.$$

Exemplo 2.11. Considere a recorrência

$$T(n) = 2.T\left(\frac{n}{2}\right) + 1,\ n > 1 \text{ e } T(1) = 0.$$

Mais uma vez, faça $n = 2^k$ e defina $G(k) = T(2^k)$. Neste caso temos, para $k > 0$

$$G(k) = T(2^k) = 2T(2^{k-1}) + 1 = 2G(k-1) + 1$$

e $G(0) = T(1) = 0$.

Assim temos

$$\begin{aligned}
G(k) &= 2G(k-1) + 1 \\
&= 2\left[2G(k-2) + 1\right] + 1 \\
&= 2^2 G(k-2) + 2 + 1 \\
&= 2^2\left[2G(k-3) + 1\right] + 2 + 1 \\
&= 2^3 G(k-3) + 2^2 + 2 + 1 \\
&= \cdots \\
&= 2^m G(k-m) + 2^{m-1} + 2^{m-2} + \cdots + 2 + 1 \\
&= \cdots \\
&= 2^k G(0) + 2^{k-1} + 2^{k-2} + \cdots + 2 + 1
\end{aligned}$$

e assim

$$G(k) = 1 + 2 + \cdots + 2^{k-1} = 2^k - 1.$$

Deste modo, como $n = 2^k$ temos

$$T(n) = n - 1.$$

Exemplo 2.12.

$$T(n) = 2.T\left(\frac{n}{2}\right) + n,\ n > 1 \text{ e } T(1) = 1.$$

Como nos exemplos anteriores, faça $n = 2^k$ e defina $G(k) = T(2^k)$. Temos agora, para $k > 0$

$$G(k) = T(2^k) = 2T(2^{k-1}) + 2^k = 2G(k-1) + 2^k$$

e $G(0) = T(1) = 1$.

Assim temos

$$\begin{aligned}
G(k) &= 2G(k-1) + 2^k \\
&= 2\left[2G(k-2) + 2^{k-1}\right] + 2^k \\
&= 2^2 G(k-2) + 2 \cdot 2^k \\
&= 2^2\left[2G(k-3) + 2^{k-2}\right] + 2 \cdot 2^k \\
&= 2^3 G(k-3) + 3 \cdot 2^k \\
&= \cdots \\
&= 2^m G(k-m) + m \cdot 2^k \\
&= \cdots \\
&= 2^k G(0) + k \cdot 2^k
\end{aligned}$$

e assim

$$G(k) = 2^k + k \cdot 2^k$$

Deste modo, como $n = 2^k$ e $k = \log_2 n$ temos

$$T(n) = (n+1)\log_2 n.$$

Voltaremos a este exemplo quando contarmos o número de comparações realizadas pelo algoritmo de ordenação merge sort, que será visto no capítulo 3.

Antes das atividades propostas que encerram o capítulo, vamos ver mais um exemplo.

Exemplo 2.13. Seja a recorrência $T(n) = n + T(n/3)$, para $n \geq 2$, e $T(1) = 1$.

Neste caso, a instância n está relacionado com a instância $n/3$, de modo que a relação só faz sentido para $n = 3^k$. Façamos então esta substituição. Encontramos assim

$$T(3^k) = 3^k + T(3^k - 1), \quad \text{e } T(3^0) = 1.$$

Assim, se $F(k) = T(3^k)$ temos

$$F(k) = 3^k + F(k-1), \quad \text{e } F(0) = 1.$$

Expandindo obtemos

$$\begin{aligned} F(k) &= F(k-1) + 3^k \\ &= F(k-2) + 3^{k-1} + 3^k \\ &= F(k-3) + 3^{k-2} + 3^{k-1} + 3^k \\ &= \cdots \\ &= F(k-m) + 3^{k-m+1} \cdot 3^k \\ &= \cdots \\ &= F(0) + 3 + 3^2 + \cdots + \cdot 3^k \\ &= 1 + 3 + 3^2 + \cdots + \cdot 3^k \\ &= \frac{3^{k+1} - 1}{2} \end{aligned}$$

Agora, como $n = 3^k$ temos

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

2.6 ATIVIDADES

1. Escreva os cinco primeiros termos das seguintes sequências:
   a) $S(1) = 10$ e $S(n) = S(n-1) + 10$, para $n \geq 2$.
   b) $T(1) = 1$ e $T(n) = n \cdot T(n-1)$, para $n \geq 2$.
   c) $W(1) = 2; W(2) = 3$ e $W(n) = W(n-1)W(n-2)$, para $n \geq 3$.
2. Encontre o termo geral a_n das sequências expressas por:
   a) $a_k - 3a_{k-1} = 0,\ a_0 = 1,\ k \geq 1$;
   b) $2a_k - 3a_{k-1} = 0,\ a_0 = 2,\ k \geq 1$.
3. Calcule uma fórmula explicita para a recorrência $F(n) = F(n/2) + 4$, com valor inicial $F(1) = 2$.
4. Mostre que $F(n) = \log_2 n + 3$, para $n = 1, 2, 2^2, 2^3, \ldots$ é uma solução da recorrência $F(n) = F(n/2) + 1$.
5. Dada a sequência de Fibonacci $F(n)$ verifique que $F(n+1) + F(n-2) = 2F(n)$, para $n \geq 2$.
6. Considere a recorrência $F(n) = 2F(n/2) + 2n$ para $n = 2^1, 2^2, 2^3, \ldots$. Resolva a recorrência adotando $F(1) = 0$.
7. Sabe-se que G é uma função definida no conjunto $\{2^0, 2^1, 2^2, 2^3, \ldots\}$ tal que $G(1) = 5$ e $G(n) = 2G(n/2) + 7n$ para $n > 1$. Determine a expressão explícita de G.
8. Considere a recorrência $F(n) = F(3n/4) + 88$. Exiba e prove uma fórmula fechada para a função F que está definida sobre as potências inteiras de $4/3$, tem $F(1) = 77$ e satisfaz a recorrência.

3

ANÁLISE DE ALGORITMOS

> *"Devemos mudar nossa atitude tradicional em relação à construção de programas. Em vez de imaginar que nossa principal tarefa é instruir o computador sobre o que ele deve fazer, vamos imaginar que nossa principal tarefa é explicar a seres humanos o que queremos que o computador faça."*
>
> — D. E. Knuth

Este capítulo será dedicado a comentar algumas aplicações de recursividade e relações de recorrência na computação. Teremos como base os problemas de ordenação e busca (Problemas 4 e 5 apresentados no Capítulo 1). Vamos analisar mais de uma estratégia de solução para cada problema, e compararemos a eficiência das mesmas.

3.1 ALGORITMOS E COMPLEXIDADE

Um algoritmo é um procedimento consistindo de um conjunto de regras que especificam uma sequência finita de operações, que produz uma solução para um problema ou classe de problemas.

Os algoritmos recursivos, nosso alvo de estudo neste capítulo, são muito usados na computação, sendo uma poderosa técnica de programação. Lembrando, são algoritmos que para resolver um dado problema, reduz o problema a uma instância menor, resolvendo ela da mesma forma, até reduzir a um caso trivial.

Um importante problema no dia a dia da computação é lidar com limitações impostas pela tecnologia, ou mesmo pelo problema atacado. Ao se escrever um programa para decidir se um corretor da bolsa deve ou não vender uma ação, é importante que este programa rode em tempo muito curto, pois estas decisões são tomadas em questão de segundos. Ao mesmo tempo, limitações tecnológicas impedem que algumas operações sejam feitas em tempo muito curto. É importante portanto que saibamos medir os recursos necessários para que um algoritmo possa resolver um problema. Tais medidas compõe o que chamamos de complexidade de um algoritmo. A complexidade de um algoritmo depende principalmente do tamanho, do formato da entrada, e do tempo de execução.

No caso de algoritmos recursivos, a complexidade de um algoritmo será representada por uma ou mais equações de recorrência. A análise de complexidade de um algoritmo é uma medida que tem parâmetros específicos do algoritmo, sendo que seu estudo depende da solução das relações de recorrência para prever a limitação do desempenho do algoritmo.

Neste trabalho nos concentraremos apenas do cálculo do tempo de execução.

3.1.1 *Análise de Complexidade*

A analise do custo de um algoritmo é feita mensurando os recursos necessários em termos de tempo de execução e de espaço. Esse tempo está ligado ao tamanho da entrada e assim procura-se definir o algoritmo mais eficiente calculando o tempo de execução do algoritmo identificando a operação básica, isto é, a operação mais custosa (cop) e tempo de execução da mesma multiplicada pela quantidade de vezes que é executada em uma entrada. Assim

$$T(n) \approx cop \cdot C(n),$$

onde $C(n)$ é o número de vezes que a operação é realizada.

Podemos exemplificar, dizendo que na multiplicação de matrizes, a adição e a multiplicação são as operações mais custosas, enquanto que em um algoritmo de ordenação, a comparação é essa operação.

3.2 COMPORTAMENTO ASSINTÓTICO

A eficiência dos algoritmos dá-se pela comparação dos mesmos em termos da chamada *comportamento assintótico* de suas funções de custo. Estamos interessados em como a função se comporta quando o tamanho da entrada é muito grande, interessando muito pouco a forma exata da função.

Antes de definir o que entendemos por comportamento assintótico, vamos considerar um exemplo. Observando as funções $f(x) = x$ e $g(x) = x^2$, podemos verificar a medida que x cresce, os valores de f e g também crescem ilimitadamente. E para x suficientemente grande teremos $g(x)$ sempre maior que $f(x)$. De fato, a diferença entre as duas aumenta a medida que x cresce, uma vez que

$$g(x) - f(x) = x^2 - x = x(x - 1),$$

que é positivo e crescente para $x > 1$.

Este fato não muda se multiplicarmos f por qualquer constante constante positiva, por maior que ela seja. Observe que

$$g(x) - cf(x) = x^2 - cx = x(x - c),$$

que é crescente e positiva para $x > c$.

Isso indica que f e g têm comportamento diferentes em relação às taxas de crescimento. Diremos neste caso que g cresce mais rápido que f.

Devido aos recursos necessários do ponto de vista computacional, a análise quanto ao trabalho requerido pelo algoritmo é tratado utilizando as seguintes medidas de complexidade:

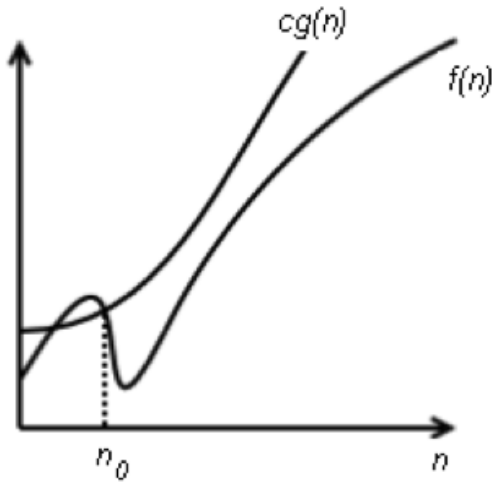
- no pior caso é previsto o uso máximo das operações básicas necessárias para se executar um dado algoritmo, sendo esta verificação mais útil para nossa análise, pois não pressupõe nada sobre o formato da entrada;
- O caso médio corresponde ao número médio de operações usadas para resolver o problema em todas suas instâncias para um determinado tamanho de entrada;
- no melhor caso a quantidade de operações é considerada a menor necessária para execução do algoritmo.

Essas medidas de complexidade podem ser feitas usando a chamada *notação assintótica*, nela se estuda o comportamento das funções para valores arbitrariamente grandes de suas variáveis, sendo chamadas de ordem de complexidade. A notação O descreve o limite superior sobre o algoritmo tratado. Para o limite inferior é utilizado a notação Ω e quando ambas tem o mesmo valor usa-se a notação Θ.

Vamos exemplificar a análise de complexidade:

Seja $g(n)$ uma função não negativa definida em $R+$.

$O(g(n))$ representa o conjunto de funções de menor ou mesma velocidade de crescimento que $g(n)$. Mais precisamente, diremos que $f(n) \in O(g(n))$ se existem constantes positivas n_0 e c tais que $f(n) \leq c \cdot g(n)$ para todo $n \geq n_0$.



As seguintes afirmações são verdadeiras:

$$n \in O(n^2);$$

$$(100n + 52) \in O(n^2);$$

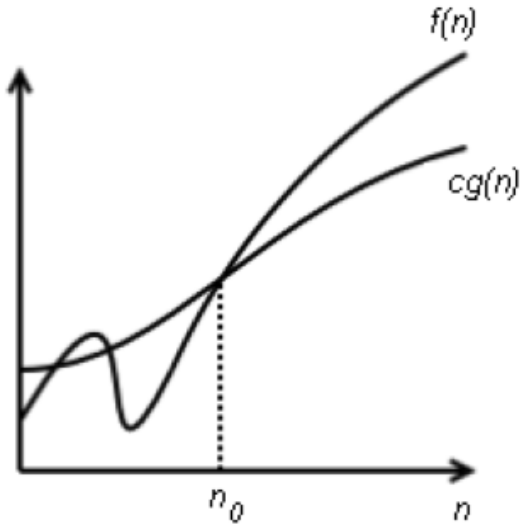
$$n(n-1) \in O(n^2)$$

Por outro lado, note que:

$$n^3 \notin O(n^2);$$

$$(n^4 + n + 1) \notin O(n^2).$$

$\Omega(g(n))$ representa o conjunto de funções de maior ou mesma ordem de grandeza que $g(n)$. Da mesma forma, diremos que $f(n) \in \Omega(g(n))$ se existem constantes positivas n_0 e c tais que $f(n) \geq c \cdot g(n)$ para todo $n \geq n_0$.



As seguintes afirmações são verdadeiras:

$$n^3 \in \Omega(n^2);$$

$$n(n-1) \in \Omega(n^2);$$

$$(200n + 10) \notin \Omega(n^2).$$

$\Theta(g(n))$ representa o conjunto de funções que tem a mesma ordem de grandeza que $g(n)$. Diremos que $f(n) \in \Theta(g(n))$ se existem constantes positivas n_0, c_1 e c_2 tais que $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ para todo $n \geq n_0$.

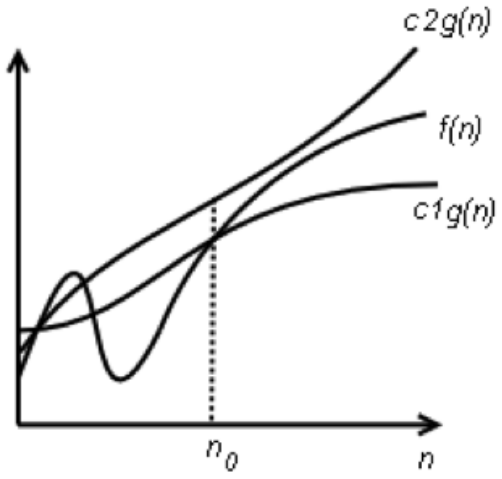
As seguintes afirmações são verdadeiras:

$$(n^2 + 3) \in \Theta(n^2);$$

$$(n^2 + \log_2 n) \in \Theta(n^2);$$

$$(200n + 10) \notin \Theta(n^2).$$

Os algoritmos recursivos podem ser escritos como uma recorrência ou utilizando procedimentos iterativos, com o qual se pode determinar a complexidade da resolução do problema.



3.3 ALGORITMOS DE ORDENAÇÃO DE LISTAS

Vamos agora tratar do problema 4 colocado no Capítulo 1. Como é fácil de perceber, a pergunta colocada não possui resposta direta e simples. A resposta depende de fato de como decidimos ordenar a lista em questão. Cada pessoa usa sua própria estratégia para fazer o ordenamento, e cada uma dessas formas acarretará em um número distinto de comparações.

Mas este não é um problema apenas teórico. Na prática, uma lista de nomes ou de números pode ser mais útil quando está ordenada, principalmente quando necessitamos acessar os dados de modo mais eficiente. Pessoas ordenam suas coleções de CD para facilitar a busca quando necessário, por exemplo.

Suponha então uma lista com n elementos, que por simplicidade pensaremos que são números, e que o nosso trabalho é ordená-los de forma crescente. Chamaremos de $C(n)$ a quantidade de comparações realizadas entre os elementos da lista.

Existem vários algoritmos que servem para colocar os elementos de uma dada sequência em ordem utilizando permutações entre os elementos da própria lista. Entre os métodos conhecidos vamos destacar e apresentar três.

3.3.1 *Ordenação por Seleção (Selection Sort)*

Este algoritmo é baseado na estratégia de localizar o menor elemento da lista e colocá-lo na primeira posição da lista, e em seguida fazer o mesmo para a lista restante.

Para isso pegamos o primeiro elemento da lista e o comparamos cada um dos elementos seguintes. Cada vez que encontramos um elemento menor do que o elemento que escolhemos, fazemos a troca, colocando o elemento maior no local onde estava o outro, e usando este novo elemento para as próximas comparações. Ao terminar, estaremos então com o menor elemento da lista. Basta agora colocá-lo no primeiro lugar da lista. Em seguida faz-se os mesmos procedimentos com os elementos não organizados da lista (que agora são $n-1$), recursivamente, até se chegar aos dois últimos elementos da lista, onde o menor deles fica como penúltimo elemento e o outro é o maior elemento da lista ficando em última posição concluindo, assim, a ordenação da lista.

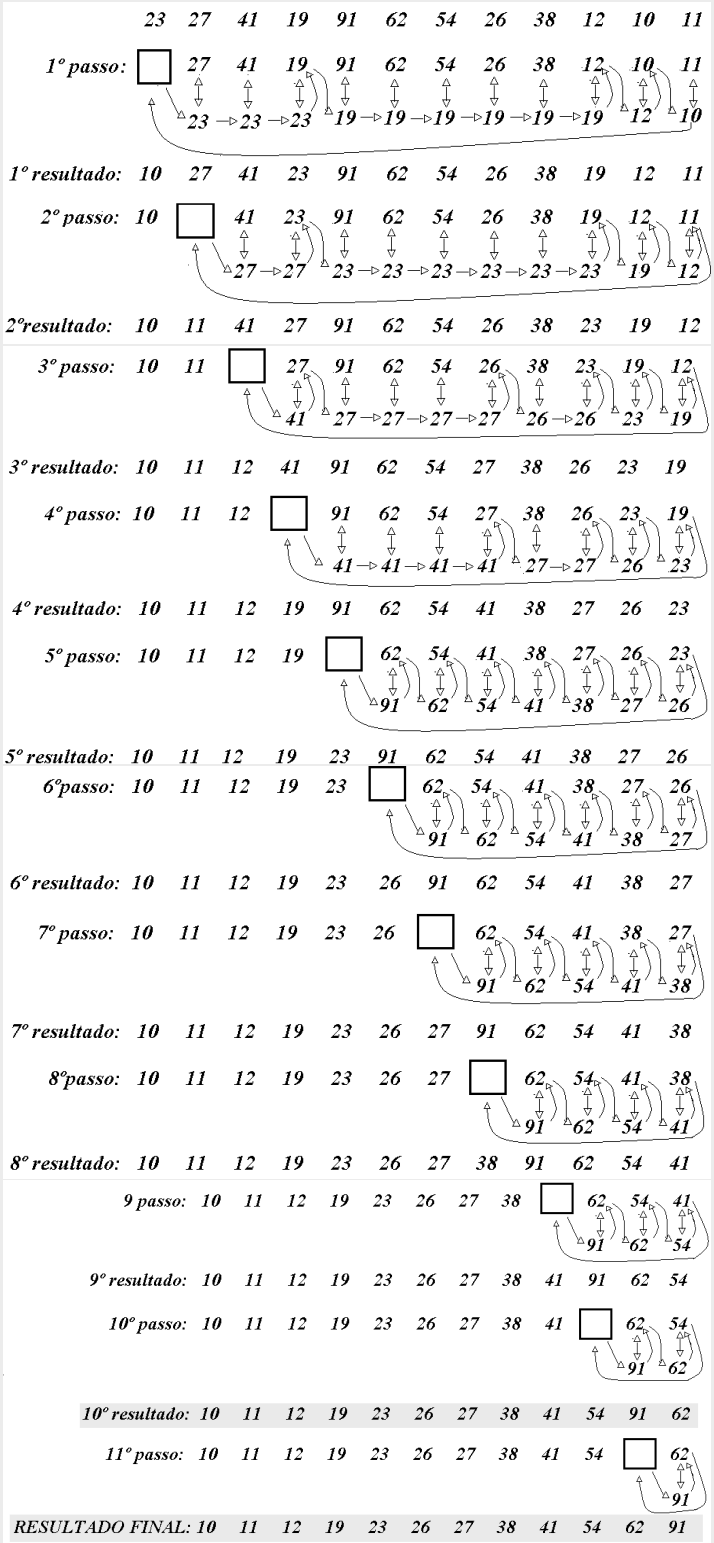
Exemplo 3.1. A seguir, uma solução para o problema de ordenação (problema 4, proposto no início do capítulo 1) usando a ordenação por seleção em uma lista com 12 elementos, sendo que o símbolo ($\updownarrow$) significa a comparação entre os elementos apresentados e a seta ($\rightsquigarrow$) mostra para onde o elemento vai após a comparação. Os procedimentos estão ilustrados na figura a seguir.

Lista: $\{23, 27, 41, 19, 91, 62, 54, 26, 38, 12, 10, 11\}$.

Os resultados: 1^o até o 10^o, são chamados de ordenação parcial enquanto o resultado final é uma ordenação completa (existem outras formas de ordenação parcial que não são relevantes a esse estudo).

Podemos verificar que nesta lista, com 12 elementos, o número de comparações foi: 11+10+9+8+7+6+5+4+3+2+1 = 66.

Assim a resposta para o problema 3 do capitulo 1, será: O número de comparações usando o método selection sort , são de 66 comparações para a ordenação da referida lista.


23 27 41 19 91 62 54 26 38 12 10 11
1° passo: 27 41 19 91 62 54 26 38 12 10 11
23 23 23 19 19 19 19 19 19 12 10
1° resultado: 10 27 41 23 91 62 54 26 38 19 12 11
2° passo: 10 41 23 91 62 54 26 38 19 12 11
27 27 23 23 23 23 23 23 19 12
2°resultado: 10 11 41 27 91 62 54 26 38 23 19 12
3° passo: 10 11 27 91 62 54 26 38 23 19 12
41 27 27 27 27 26 26 23 19
3° resultado: 10 11 12 41 91 62 54 27 38 26 23 19
4° passo: 10 11 12 91 62 54 27 38 26 23 19
41 41 41 41 27 27 26 23
4° resultado: 10 11 12 19 91 62 54 41 38 27 26 23
5° passo: 10 11 12 19 62 54 41 38 27 26 23
91 62 54 41 38 27 26
5° resultado: 10 11 12 19 23 91 62 54 41 38 27 26
6°passo: 10 11 12 19 23 62 54 41 38 27 26
91 62 54 41 38 27
6° resultado: 10 11 12 19 23 26 91 62 54 41 38 27
7° passo: 10 11 12 19 23 26 62 54 41 38 27
91 62 54 41 38
7° resultado: 10 11 12 19 23 26 27 91 62 54 41 38
8°passo: 10 11 12 19 23 26 27 62 54 41 38
91 62 54 41
8° resultado: 10 11 12 19 23 26 27 38 91 62 54 41
9 passo: 10 11 12 19 23 26 27 38 62 54 41
91 62 54
9° resultado: 10 11 12 19 23 26 27 38 41 91 62 54
10° passo: 10 11 12 19 23 26 27 38 41 62 54
91 62
10° resultado: 10 11 12 19 23 26 27 38 41 54 91 62
11° passo: 10 11 12 19 23 26 27 38 41 54 62
91
RESULTADO FINAL: 10 11 12 19 23 26 27 38 41 54 62 91

Considere uma lista de n elementos e denote por $C(n)$ o total de comparações feitas pelo algoritmo para ordenar esta lista.

Observe agora que em uma lista com n elementos são feitas inicialmente $(n-1)$ comparações para colocar o menor elemento na primeira posição. Feito isso resta ordenar os outros $n-1$ elementos, e necessitará um total de $C(n-1)$ comparações. No caso onde $n = 1$ (a lista tem apenas um elemento) não é necessário fazer comparações. Concluímos assim que

$$C(n) = C(n-1) + (n-1)$$

$$C(1) = 0.$$

Abrindo temos que

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

Neste caso diz-se que a complexidade desse algoritmo é de ordem quadrática. Ou seja, $C(n) \in O(n^2)$, pois esses resultados crescem de forma similar.

É interessante observar que neste método não existe diferença entre as situações pior, média ou melhor, pois são realizadas todas as comparações necessárias partindo do primeiro elemento não ordenado.

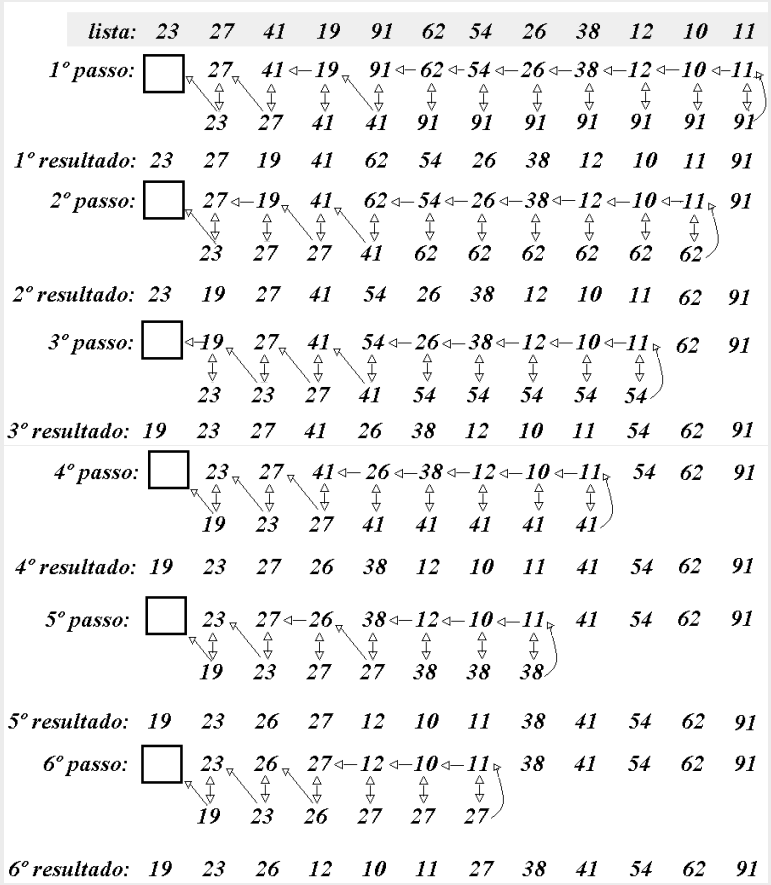
3.3.2 *Ordenação por Flutuação (Bubble Sort)*

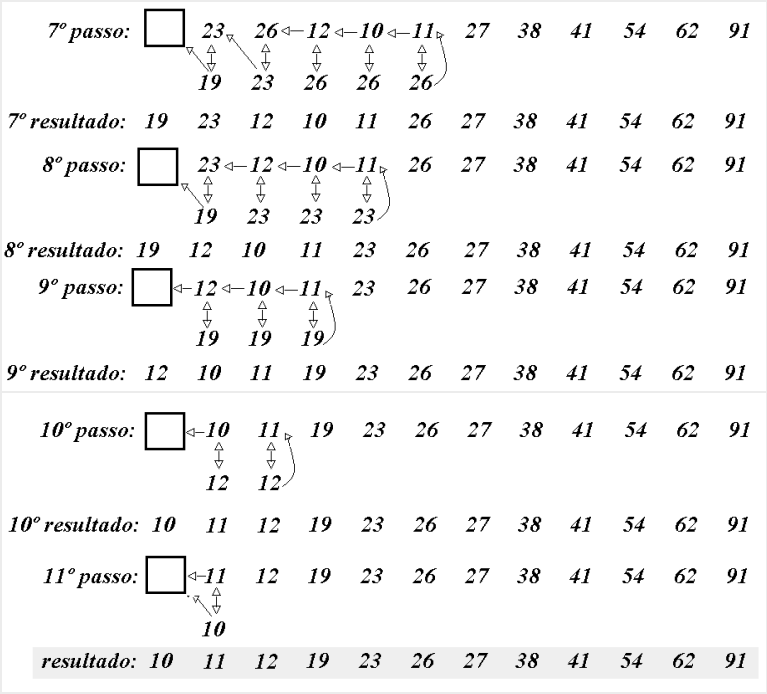
Este método de ordenação tem uma estratégia similar ao algoritmo anterior, mas com uma técnica distinta. A ideia é localizar o maior elemento e levá-lo ao final da lista. Para fazer isso a ideia é comparar o primeiro elemento com o segundo, colocando-os em ordem crescente, em seguida comparar o segundo com o terceiro elemento colocando-os em ordem crescente, na segunda e terceira posição da lista, fazendo assim até comparar (e se necessário trocar) os dois último elementos da lista.

Note que ao passarmos pelo maior elemento da lista, este elemento ganhará todas as comparações subsequentes, forçando a troca e passando para a comparação seguinte. Com isso, ao final do processo, o maior elemento estará no final da lista.

Inicia-se novamente o processo nos $n - 1$ elementos ainda não ordenados, recursivamente, até restar apenas um elemento (caso trivial).

Exemplo 3.2. Agora ilustraremos a aplicação do método de ordenação por bolha na mesma lista apresentada no problema 4 do capítulo 1. Ao final poderemos observar que a resposta ao problema utilizando o método selection sort ou o método bubble sort necessitam de 66 comparações para a ordenação da lista dada.





Considere agora uma lista de n elementos. Assim como no algoritmo de ordenação por seleção, a ordenação por flutuação faz na primeira passagem um total de $n-1$ comparações (todos os pares consecutivos de posições), e em seguida ordena recursivamente os elementos restante. Assim, se $C(n)$ é o total de comparações, então

$$C(n) = C(n-1) + (n-1)$$

$$C(1) = 0.$$

e

$$C(n) = \frac{n^2 - n}{2},$$

assim como no selection sort.

A vantagem deste algoritmo está na possibilidade de pararmos os algoritmo antes. Note que se ao passar fazendo as comparações em cada par de posições consecutivas, nenhum par de elementos trocar de posição é por que a lista já está ordenada! Com isso podemos colocar um gatilho em nosso algoritmo para terminar a execução quando, em alguma instância, nenhum troca de posições for feita.

Assim, se a lista já estiver ordenada deste o início, o algoritmo fará apenas $n - 1$ comparações.

A complexidade deste método é portanto, no pior caso é $O(n^2)$, enquanto que no melhor caso é $O(n)$.

3.3.3 *Ordenação por Intercalação (Merge Sort)*

Criado por John Von Neumann em 1945, o merge sort, ou ordenação por intercalação, é um exemplo de algoritmo de ordenação do tipo "dividir para conquistar". Estratégia esta que consiste em dividir o problema de forma balanceada (gerando subproblemas de mesmo tamanho) para conquistar o objetivo. [11]

A idéia do merge sort é dividir o problema em duas listas de mesmo tamanho (com uma diferença de um elemento se o total de elementos é ímpar), ordená-los recursivamente, e depois fundir as duas listas de modo ordenado.

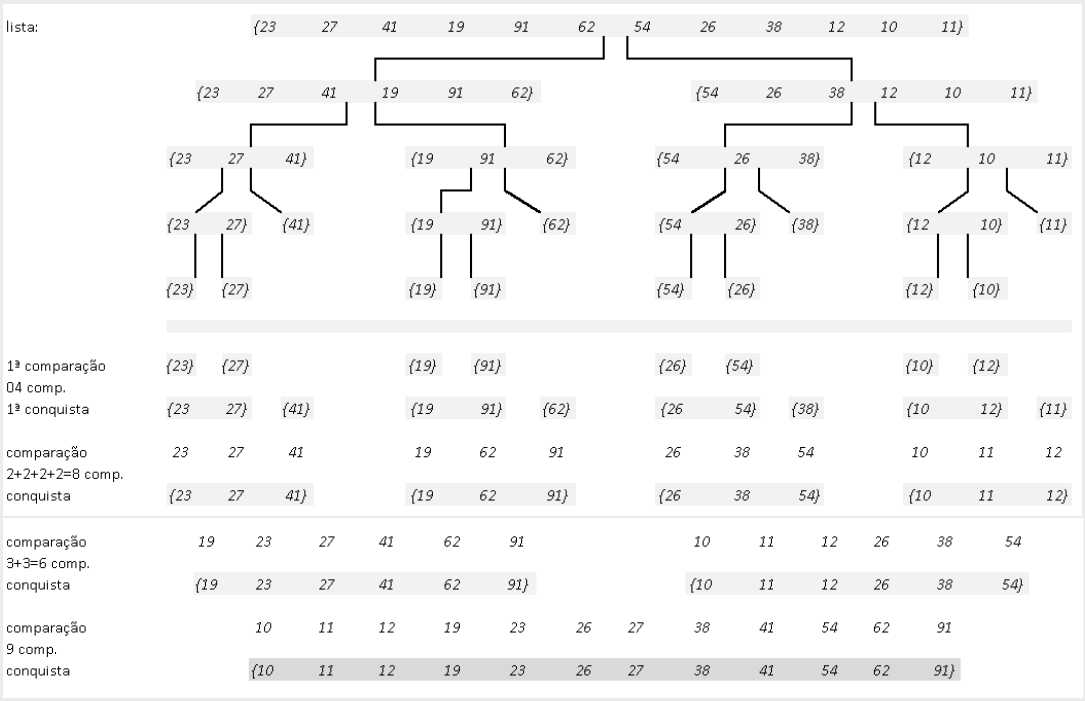
Abrindo a recursão, podemos dividir este método em três etapas para efetuar a ordenação de uma lista. (ver [11]):

1. Divisão dos dados da lista em duas listas menores , sendo que estas listas se dividem até que cada lista tenha exatamente um elemento;
2. Faz-se as comparações entre cada dupla de elementos que originaram as listas finais;
3. Refazer as listas da etapa imediatamente anterior combinando as duas metades em uma lista organizada.

Uma parte essencial deste algoritmo está em como realizar a fusão das listas ordenadas. Para isso, ele começa pegando o primeiro elemento da primeira lista como elemento de comparação. Sabemos que todos os demais elementos da primeira lista são maiores que o escolhido, de modo que se um elemento da outra lista que está sendo comparada é menor que ele, então é menor que todos os da lista 1. Começamos então comparando o elemento escolhido com os elementos da lista 2. Se o elemento da lista 2 for menor que o escolhido ele é colocado em uma lista auxiliar, na primeira posição disponível. Isso segue até encontrarmos um elemento maior. Neste momento o elemento de comparação vai para a lista auxiliar, e o elemento

seguinte da lista 1 (maior) toma o lugar dele, como elemento de comparação. Este procedimento segue até que todos os elementos estejam na lista auxiliar, que estará ordenada.

Exemplo 3.3. Aplicando este método na lista utilizada pelo problema 4 do capítulo 1, tem-se:



Observamos que neste método, a quantidade de comparações foi:

$$4 + 8 + 6 + 9 = 27,$$

ou seja, neste método efetuou-se menos comparações, em relação aos métodos utilizados anteriormente, podendo ser visto como uma melhor opção que as anteriores.

Funcionamento para uma lista com n elementos

Antes de mais nada é importante localizar o pior e melhor caso deste algoritmo. Podemos definir o que é melhor e pior caso para a ordenação por mistura da seguinte maneira.

Melhor Caso - nunca é necessário trocar elementos após comparações.

Pior Caso - sempre é necessário trocar elementos após comparações.

Vamos nos concentrar na análise do pior caso, e como o algoritmo sempre divide a lista pela metade, vamos considerar $n = 2^k$. Como estamos interessados no comportamento assintótico da função, este caso é o suficiente para nossa análise.

Assim, no pior caso, ao fazer a fusão de duas listas de tamanho $n/2$, como o algoritmo sempre troca de elemento ao comparar, o total de comparações feitas é n (deixamos como exercício a verificação dessa afirmação.)

Para ordenar uma lista de tamanho n o merge sort divide a lista em duas de tamanho $n/2$, ordena cada uma delas recursivamente, e funde as duas.

Desta maneira, o o total de comparações do Merge sort no pior caso é dado por

$$C(n) = \begin{cases} 1 & , \quad \text{se } n = 1; \\ 2C\left(\frac{n}{2}\right) + n & , \quad \text{se } n \geq 2; \end{cases} \tag{3.1}$$

Esta equação já foi resolvida no exemplo 2.12, onde encontramos que

$$C(n) = (n+1)\log_2 n,$$

e portanto $C(n) \in O(n \log_2 n)$.

Verificando o funcionamento para n elementos, temos:

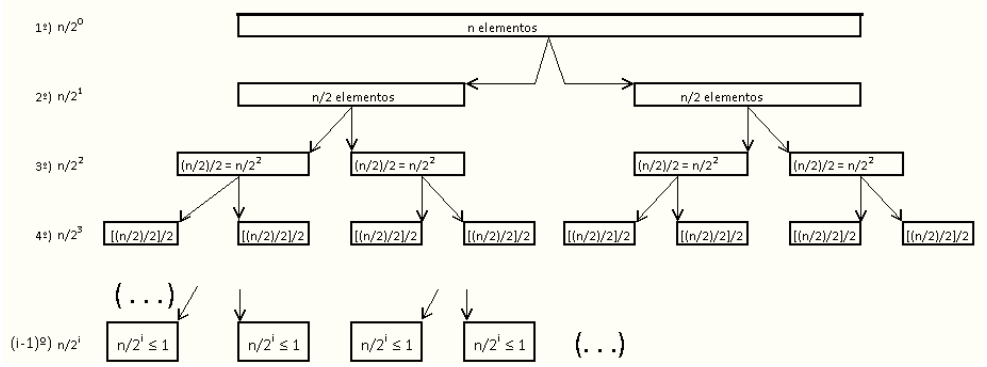


Figura 13: Exemplo de funcionamento do algoritmo de ordenação por intercalação

3.4 ALGORITMOS DE BUSCA

É um algoritmo, utilizado em computação com a finalidade de fazer uma busca para encontrar um registro ou elemento desejado, chamado de chave, em uma lista (pode ser uma tabela, vetor de registros, uma árvore ...). A operação de busca é uma das três operações mais frequentes numa lista, junto com a inserção e a remoção de um elemento da lista.

As técnicas de busca que mostraremos são as de busca sequencial e busca binária.

3.4.1 *Busca Sequencial*

É a forma mais simples de busca. Ela é aplicável a uma tabela ordenada ou não, porém a eficiência da operação de busca melhora se os registros estiverem ordenados, pois podemos parar a busca quando encontramos um registro maior que o que buscamos.

FUNCIONAMENTO: A chave procurada é comparada com cada um dos elementos da lista, iniciando pelo primeiro da lista até encontrar o elemento procurado ou até chegar ao final dos elementos da lista, retornando com o resultado da procura.

O algoritmo tem um tempo de processamento máximo quando a chave é o último item da lista, ou quando não é um elemento da lista. Nos dois casos, todos os elementos da lista são comparados a chave, totalizando assim n comparações. É claro que a chave poderia ser o primeiro item da lista, o que acarretaria apenas uma comparação; no entanto, se a chave estiver no meio da lista serão necessárias aproximadamente $n/2$ comparações. Obviamente, existem muitas possibilidades e seria interessante que tivéssemos alguma forma de estimar o valor médio da quantidade de processamento demandada. Tal estimativa requer informações sobre a lista típica sobre a qual se realiza a busca, bem como uma relação típica entre o valor procurado e a lista dada. Porém, uma medida para o comportamento médio é geralmente muito difícil de sua determinação, não só para este algoritmo específico, mas para a maioria dos algoritmos.

Pesquisa com sucesso:

- melhor caso: $C(n) = 1$;
- pior caso: $C(n) = n$;
- caso médio: $C(n) = (n+1)/2$.

Pesquisa sem sucesso: $C(n) = n + 1$ no pior caso.

O algoritmo de pesquisa sequencial é a melhor escolha para o problema de pesquisa em tabelas com até 25 registros. [12]

3.4.2 *Busca Binária*

É um algoritmo aplicado somente quando os dados estiverem ordenados na lista e não permite a existência de duplicidade (elementos duplicados) na lista. A busca binária é quase sempre mais eficiente que a busca sequencial. Entretanto, um algoritmo de busca sequencial tem uma grande vantagem: se a lista a ser pesquisada não estiver ordenada, o algoritmo de busca sequencial funcionará, mas o de busca binária, não. Devemos ordenar a lista para então usarmos o algoritmo de busca binária, assim, devemos considerar o número de operações envolvidas no procedimento de ordenação.

FUNCIONAMENTO DA BUSCA BINÁRIA: Para saber se uma chave está presente na tabela:

1. Compare o elemento procurado (chave) com o registro que está na posição do meio da lista;
2. Se a chave é menor então o registro procurado está na primeira metade da lista;
3. Se a chave é maior então o registro procurado está na segunda metade da lista;
4. Repita até que a chave seja encontrada ou que se constate que a chave não existe na lista.

Exemplo 3.4. Tomemos como exemplo o problema 5 do capítulo 1, supondo que o jogador A tenha escolhido o valor 356.

Ou seja, estamos buscando o elemento 356 na lista de números de 1 a 1000.

Solução:

1. Como o número deve estar entre 1 e 1000, o resultado escolhido será 1000/2=500, sendo que será dito que o número chave é menor que o valor dado.
2. Agora temos que o número está entre 1 e 500. Assim, o método diz para escolher 500/2=250. Como resposta teremos que a chave é um número maior que o valor apresentado.
3. Temos agora que a chave está entre 250 e 500, logo pela busca binária temos que (250+500)/2= 375, que é o valor escolhido pelo método. Como resposta teremos que a chave é um número menor que o valor apresentado.
4. Agora, pela busca temos que o número a ser passado está entre 250 e 375, ou seja, (250+375)/2= 312,5 (neste caso usaremos o 312) e será dito que o número continua sendo maior que esse valor.
5. Continuando, temos que: (312+375)/2=343,5 (usamos 343) e a chave ainda é um número maior.
6. Agora, temos: (343+375)/2=359 e nos será dito que a chave é um número menor.
7. Logo, (343+359)/2= 351, onde a chave é um número maior que este.
8. Então, (351+359)/2= 355 e novamente seremos informados que a chave é maior.
9. Agora, (355+359)/2= 357 é o número dado pela busca binária e ainda não chegamos ao resultado procurado.
10. Temos aqui: (355+357)/2= 356, que é o resultado procurado. Assim, verificando a quantidade de operações, temos que foram necessárias dez chutes para chegar ao resultado correto.

A Figura 14 a seguir mostra o exemplo anterior em forma de diagrama esquemático.

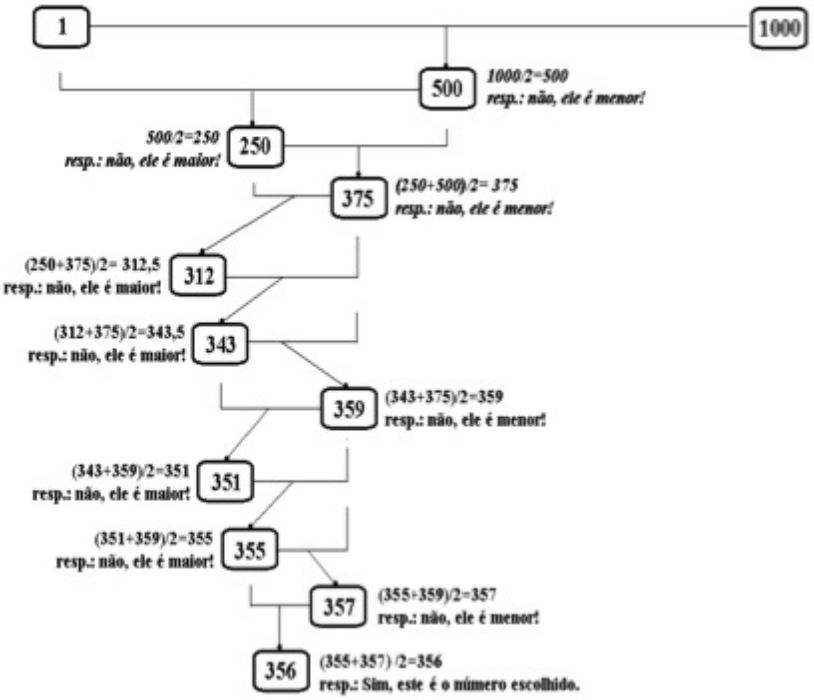


Figura 14: Diagrama esquemático da busca realizada no exemplo 3.4.

ANÁLISE DA COMPLEXIDADE: A primeira coisa a fazer é localizar o pior caso do algoritmo. Não é difícil perceber que a pior situação é aquela em que temos que reduzir o tamanho das listas ao mínimo para encontrar o elemento procurado, assim como no exemplo acima.

Neste caso o algoritmo escolhe o número médio, faz uma comparação e escolhe uma lista de tamanho $n/2$, onde realiza uma busca nos mesmos moldes. Assim

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + 1, & \text{se } n \geq 2 \\ 1, & \text{se } n = 1. \end{cases}$$

Assim como no caso do merge sorte consideraremos, sem perda de generalidade que $n = 2^k$. Fazendo então esta substituição temos

$$T(2^k) = \begin{cases} T(2^{k-1}) + 1, & \text{se } k \geq 1 \\ 1, & \text{se } k = 0. \end{cases}$$

Segue facilmente que

$$T(2^k) = kT(2^0) = k,$$

e como $k = \log_2 n$, então

$$T(n) = \log_2 n.$$

3.5 ATIVIDADES

1. O algoritmo de busca binaria é utilizado na lista a seguir; x tem o valor "Cuiabá". Enumere os elementos com os quais x é comparado.

   (Brasília, Campinas, Ipu, Nova Iguaçu, Petrolina, São Paulo, Xerém)

2. Um algoritmo de busca binária é usado na lista a seguir, tendo como valor de busca x = "margarina". Indique com quais elementos x é comparado.

   (açúcar, chocolate, manteiga, margarina, nata, ovos)

3. Em cada passo do algoritmo selection sort, o índice do item de valor máximo em uma lista deve ser determinado. Isto requer comparações entre os elementos da lista. Quantas comparações são necessárias no pior caso para determinar o maior elemento de uma lista não ordenada com 17 elementos?

4. A parte do algoritmo merge sort que executa o merge requer a comparação entre elementos das duas listas ordenadas a fim de escolher qual elemento será incluído na lista ordenada, que será a combinação das duas "metades já ordenadas". Quando o fim de uma das listas é alcançado, os elementos restantes da outra lista podem ser adicionados à nova lista sem a necessidade de mais comparações. Dadas as duas listas, nos itens abaixo, execute um merge e conte o número de comparações necessárias para executá-lo.

   a) $5, 7, 9$ e $1, 3, 4$

   b) $1, 6, 7$ e $2, 3, 5$

   c) $0, 1, 3, 5, 7, 11$ e $2, 8, 9, 10, 12$

BIBLIOGRAFIA

[1] WIKIPEDIA. Wikipedia - M.C.Escher. http://pt.wikipedia.org/wiki/Maurits_Cornelis_Escher, 2013.

[2] SCHATTSCHNEIDER, D. The mathematical side of m. c. escher. *Notices of the AMS*, v. 57, n. 6, p. 706–718, Junho/Julho 2010.

[3] FOUNDATION, M. Site Oficial de M.C.Escher. http://www.mcescher.com, 2013.

[4] MARTA; FELIPE. Trabalho da disciplina "Seminário Temático" - Universidade de Lisboa. http://www.educ.fc.ul.pt/docentes/opombo/seminario/escher/browser.html, 2002.

[5] LUCAS, É. *Jeux scientifiques pour servir à l'histoire, à l'enseignement et à la pratique du calcul et du dessin- la tour d'hanoi*. 3. Paris: Chambon & Baye, 1889.

[6] C.R.MORAES. *Estruturas de dados e algoritmos, uma abordagem didática*. São Paulo: Editora Berkeley, 2001.

[7] DE GLASMAN, J. B. Revista Morasha - *Flávio Josefo, entre Roma e Jerusalém*. http://www.morasha.com.br/conteudo/artigos/artigos_view.asp?a=862&p=4, setembro 2010.

[8] WIKIPEDIA. Wikipédia - Liber Abaci. http://pt.wikipedia.org/wiki/Liber_Abaci, 2013.

[9] BOYER, C. B. *História da matemática*. São Paulo: Edgar Blucher, 1974.

[10] PAIVA, R. E. B. Progressões aritmético-geométricas (pag) e progressões geométrico-aritméticas (pga). *Revista do Professor de Matemática*, , n. 73, p. 47–49, setembro 2010.

[11] GRAHAM, R. L.; KNUTH, D. E.; PATASHINIK, O. *Concrete mathematics*. New York: Addison-Wesley Publishing Company, 1990.

[12] TOFOLLO, T. Notas de aula de bcc202 - aula 21. Disponível em http://www.decom.ufop.br/toffolo/site_media/uploads/2011-1/bcc202/slides/21._pesquisa.pdf.

[13] POLLMAN, H. S. Equações de recorrência. *Revista Eureka!*, , n. 9, p. 33, 2000.

[14] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos - teoria e prática*. 2. ed. Rio de Janeiro: Editora Campus, 2002.

[15] SZWARCFTITER, J. L.; MARKERNZON, L. *Estruturas de dados e seus algoritmos*. Rio de Janeiro: LTC Editora S.A., 1994.

[16] GERSTING, J. L. *Fundamentos matemáticos para a ciência da computação*. 3. ed. Rio de Janeiro: LTC Editora S.A., 1993.

[17] PARDO, T. A. S. Métodos de Busca - Slides de curso. http://wiki.icmc.usp.br/images/7/7c/M%C3%A9todos_de_busca_-_parte1_2010.pdf, 2008.

[18] ROBERTSON, J. O. . E. F. Mactutor history of mathematics - *François Edouard Anatole Lucas*. http://www-history.mcs.st-andrews.ac.uk/Biographies/Lucas.html, dezembro 1996.

[19] MIYAZAWA, F. K. Notas de aula de complexidade de algoritmo. Disponível em http://www.dcc.ufrj.br/~francisco_vianna/livros/Apostila.de.Algoritmos.-UNICAMP.pdf, 2013.