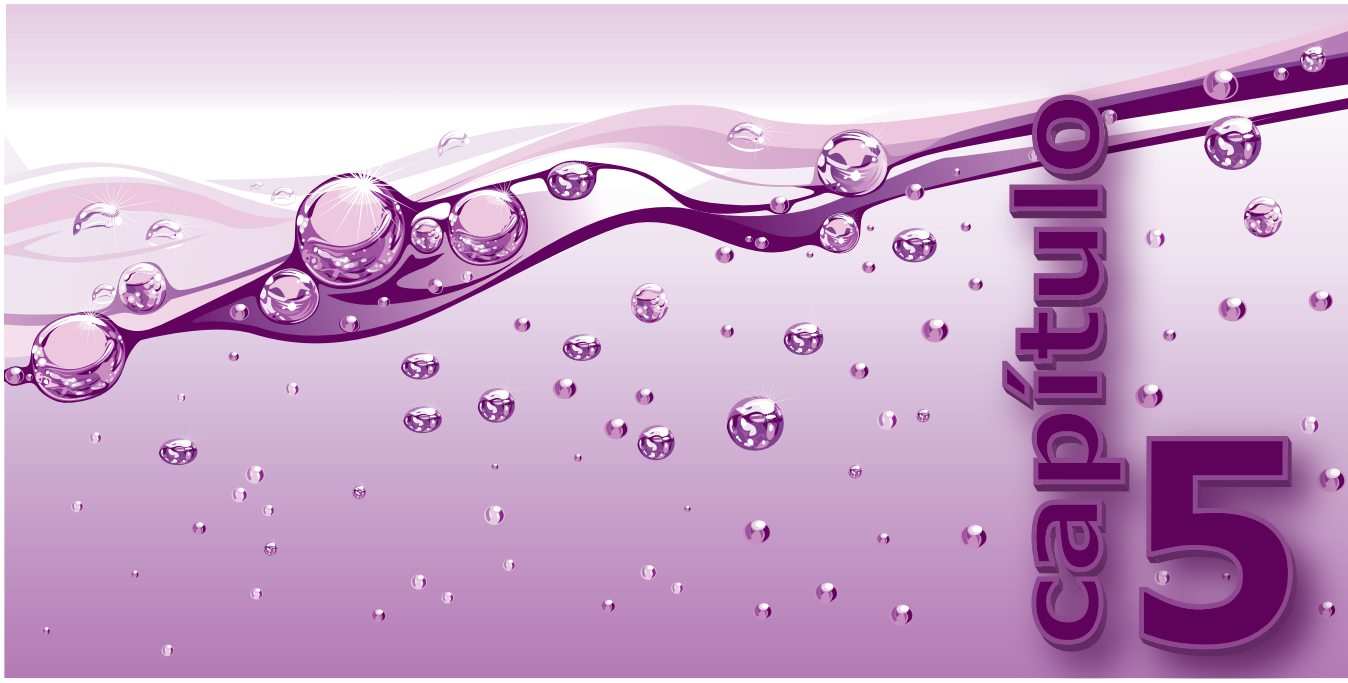


Análise Amortizada

objetivos

Após estudar este capítulo, você deverá ser capaz de:

- Definir e usar os seguintes conceitos no contexto de análise amortizada de algoritmos:
  - ❒ Amortização
  - ❒ Custo amortizado
  - ❒ Moeda virtual
  - ❒ Método de agregado
  - ❒ Método contábil
  - ❒ Método de potencial
  - ❒ Soma telescópica
  - ❒ Função potencial
- Discutir a principal motivação que norteia análise amortizada de algoritmos
- Descrever os métodos de análise amortizada
- Explicar por que o tamanho de um array dinâmico deve crescer geometricamente
- Apresentar as diferenças e semelhanças entre análise amortizada e análise assintótica
- Decidir se análise amortizada é adequada na avaliação de determinado algoritmo
- Avaliar algoritmos usando análise amortizada

ANÁLISE ASSINTÓTICA, discutida em detalhes no **Capítulo 6** do **Volume 1**, é uma ferramenta matemática poderosa e tem sido usada desde então na análise de custos de algoritmos e estruturas de dados. Contudo existem situações em programação nas quais a análise assintótica rigorosa, apesar de correta, resulta em custos considerados pessimistas demais.

Considere, por exemplo, a tabela de busca implementada usando array dinâmico discutida na **Seção 3.3.2**. Naquela ocasião foi afirmado que o custo temporal de uma operação de inserção nessa tabela de busca era $\theta(n)$ devido ao fato de uma tal operação requerer um redimensionamento do array, que tem custo temporal $\theta(n)$ no pior caso. Agora suponha que o programa que processa essa tabela aloca um espaço capaz de conter um milhão de elementos e que o referido array dobra de tamanho sempre que é redimensionado. Então, se houver uma sequência de um milhão e meio de chaves inseridas nessa tabela, apenas uma dessas inserções será de fato dispendiosa, que é aquela que requer redimensionamento do array; todas as demais inserções terão custo temporal $\theta(1)$. Portanto não parece ser justo que cada uma dessas inserções seja avaliada com custo $\theta(n)$.

Este capítulo estuda em detalhes o problema de tabela de busca implementada usando array dinâmico na **Seção 5.3** usando uma ferramenta denominada **análise amortizada de algoritmos**. A ideia básica que norteia esse tipo de análise é o exame de operações em conjunto (e não individualmente), de modo que as poucas operações dispendiosas quando combinadas com muitas operações menos onerosas resultem numa boa avaliação de desempenho quando se considera uma longa sequência de operações.

Como foi visto no **Capítulo 4**, uma árvore afunilada pode se tornar muito desbalanceada, fazendo com que um único acesso a um nó da árvore seja muito dispendioso. Mas, neste capítulo, será mostrado que, ao longo de uma sequência de acessos, árvores afuniladas não são tão dispendiosas e não requerem muito mais operações de rotação do que árvores AVL, por exemplo. Uma avaliação de cada operação básica sobre árvore afuniladas usando análise amortizada será apresentada na **Seção 5.4**.

5.1 Fundamentos

5.1.1 Conceitos

Análise amortizada é uma importante ferramenta de análise de algoritmos que facilita o entendimento dos custos temporais de estruturas de dados que apresentam operações com desempenhos variados.

O **custo amortizado** de uma sequência de m operações é o custo total dessa sequência dividido por m. Ou seja, quando uma sequência de m operações apresenta custo total igual a $\theta(m \cdot f(n))$, o custo amortizado de cada operação é igual a $\theta(f(n))$. Durante a execução dessas m operações, algumas delas podem ter custo maior do que $\theta(f(n))$, mas, em compensação, algumas outras operações podem ter custo bem menor do que esse.

Em vez de enfocar em cada operação isoladamente como ocorre em análise assintótica tradicional, análise amortizada considera interações entre operações que constituem uma **sequência de operações**. A análise amortizada é motivada pelo fato de a análise assintótica de pior caso por operação poder resultar num limite pessimista demais quando a única situação que suscita uma operação onerosa é a ocorrência de um grande número de operações módicas precedentes.

Para muitas estruturas de dados, tal como a tabela dinâmica mencionada no início deste capítulo, não é razoável considerar o custo de pior caso para cada operação numa sequência de operações, de forma que, nesse caso, a análise assintótica pode resultar num custo muito pessimista. Levando em conta as propriedades do problema em questão e os algoritmos envolvidos em sua solução, a análise amortizada permite encontrar um custo que reflete melhor o desempenho desses algoritmos, pois ela leva em consideração o desempenho médio de cada operação no pior caso.

A ideia básica que norteia a análise amortizada é que uma operação de alto custo pode alterar o estado de uma estrutura de dados de tal maneira que essa operação não ocorra novamente por um longo período, o que amortiza seu custo. Ou seja, análise amortizada é usada para algoritmos nos quais uma operação ocasional é muito lenta, mas muitas outras operações são bem mais rápidas. A análise amortizada tenta garantir que o custo médio de uma sequência de operações, no pior caso, é menor do que o custo no pior caso de uma operação onerosa específica.

Tanto análise assintótica quanto análise amortizada devem ser consideradas quando se escolhe um algoritmo para uso na prática, pois, como a análise amortizada não informa nada com respeito ao custo de operações isoladas, é possível que uma operação de uma sequência de operações tenha um custo enorme.

A **Tabela 5–1** resume as situações nas quais a análise amortizada deve ou não ser usada.

Quando Usar análise amortizada	Quando Não Usar análise amortizada
A operação que representa o pior caso do algoritmo ocorre esporadicamente e, quando ela ocorre, é compensada por um número muito maior de operações de baixo custo	Operações de alto custo não são compensadas por operações de baixo custo
O programa que executa o algoritmo suporta uma eventual operação com alto custo	É importante para o programa que executa o algoritmo que todas as operações tenham custos baixos

Tabela 5–1: Usos de Análise Amortizada

5.1.2 Comparação com Análise Assintótica de Caso Médio

Tipicamente, em análise de algoritmos tradicional, são considerados três casos: melhor caso, pior caso e caso médio. Em cada um desses casos, considera-se uma única execução do algoritmo e tenta-se determinar qual é o seu custo para processar a respectiva entrada. Análise amortizada difere de análise assintótica por apresentar uma estimativa de custo de pior caso para uma longa sequência de operações, em vez de para uma única operação isoladamente. Ou seja, análise amortizada avalia o mesmo pior caso da análise tradicional, mas ela leva em conta a ocorrência dele em conjunto com outras operações que não são tão ruins.

Análise amortizada é semelhante à análise assintótica de caso médio, pois ela é associada ao custo médio de uma sequência de operações. Entretanto análise tradicional de caso médio conta com suposições estatísticas sobre os dados de entrada do algoritmo cujo custo se deseja estimar (v., por exemplo, o **Teorema 3.3**). Portanto sua validade depende de suposições sobre distribuições desses dados, o que significa que essa análise é inválida se essas suposições não forem válidas. Por outro lado, análise amortizada não usa tais suposições.

5.2 Métodos de Análise Amortizada

Existem três métodos básicos usados em análise amortizada[1]:

- **Método de agregado**, no qual qualquer operação que constitui uma sequência de operações sob análise contribui com o mesmo peso para o resultado final da análise.
- **Método contábil**, que é baseado num modelo financeiro que impõe uma cobrança extra na execução de operações módicas e usa o excedente para compensar execuções de operações dispendiosas.
- **Método de (energia) potencial**, que é baseado num modelo de energia da Física no qual é usada uma função potencial que caracteriza a quantidade de energia que se tem à disposição para executar uma

[1] A denominação do método agregado é derivada de um dos significados da palavra *aggregate* em inglês, que é um todo constituído por várias partes que, tipicamente, são díspares. Em português, a palavra *agregado* não tem exatamente esse significado.

determinada operação. Os valores dessa função aumentam ou diminuem de acordo com a execução de cada operação sucessiva e não podem ser negativos.

Para executar uma análise amortizada, deve-se escolher um desses métodos. Essas abordagens produzem resultados equivalentes, mas uma delas pode ser mais apropriada do que outra para um determinado problema. Não existe nenhuma fórmula mágica para a obtenção de uma função de potencial ou esquema contábil que sempre funcione.

5.2.1 Método de Agregado

O método de agregado é uma abordagem simples que consiste em calcular o custo para uma sequência de operações e depois dividir esse custo pelo número de operações para obter o custo amortizado de cada operação. O mesmo custo amortizado é atribuído a cada operação, mesmo quando essas operações possuem naturezas distintas (p. ex., qualquer inserção numa tabela dinâmica tem o mesmo custo amortizado quer ela requeira redimensionamento da tabela ou não). Na prática, esse método não é muito aplicável, de modo que ele é usado em conjunto com um dos demais métodos de análise amortizada.

5.2.2 Método Contábil

O método contábil de análise amortizada usa um esquema de créditos e débitos para acompanhar os custos de diferentes operações consideradas em sequência. Para levar a efeito essa abordagem, considera-se o computador como uma máquina que requer o pagamento de uma **moeda virtual** por uma quantia constante de tempo de processamento. Quando uma operação é executada, deve haver moedas virtuais disponíveis suficientes para pagar por seu custo temporal.

Deve-se enfatizar que o método contábil é simplesmente uma ferramenta de análise. Ele não requer que se modifique uma estrutura de dados ou um algoritmo de qualquer modo. Por exemplo, esse método não requer que se acrescentem componentes para armazenar moedas virtuais gastas ou economizadas. Além disso, a cobrança não corresponde necessariamente ao tempo real requerido para uma operação específica. Quer dizer, é possível que uma operação termine em menos tempo do que o tempo que é cobrado pela operação. Em tal caso, sobra um saldo positivo para cobrir operações futuras.

Pode-se fazer uma analogia entre o método contábil e a manutenção de uma conta bancária na qual operações de baixo custo são cobradas um pouco mais do que seus custos reais e o excedente é depositado nessa conta para uso posterior. Operações de custo elevado podem então ser cobrados menos do que seu custo real e o déficit é pago pelo que foi poupado na conta. Desse modo, repartem-se os custos de operações mais onerosas com a sequência inteira. O valor cobrado por cada operação deve ser suficientemente elevado para que o saldo na conta bancária sempre permaneça positivo, mas, ao mesmo tempo, suficientemente baixo para que não seja cobrado de nenhuma operação muito além do que ela realmente custa. Em resumo, o segredo para uma análise amortizada bem-sucedida utilizando o método contábil é efetuar cobranças apropriadas e mostrar que elas são suficientes para cobrir o custo de qualquer sequência de operações.

Suponha que se tenha uma sequência de m operações sobre uma estrutura de dados e seja r_i o custo real da operação i para $1 \leq i \leq m$. Suponha ainda que os valores da função de crédito sejam c_0, c_1, ..., c_m, sendo c_0 o saldo antes da primeira operação e c_i o saldo depois da operação i, para $1 \leq i \leq m$. Assim o custo amortizado a_i de cada operação é definido como:

$$a_i = r_i + c_i - c_{i-1} \qquad \textbf{Equação 5–1}$$

A **Equação 5–1** significa que o custo amortizado é o custo real mais a alteração de saldo que ocorreu durante a execução da operação. Essa equação pode ser rearranjada como $r_i = a_i + c_{i-1} - c_i$, de forma que o custo real total é dado pelo **Lema 5.1**.

Lema 5.1: O custo real total e o custo amortizado total de uma sequência de m operações numa estrutura de dados são relacionados por:

$$\sum_{i=1}^{m} r_i = \sum_{i=1}^{m} (a_i + c_{i-1} - c_i) = \left(\sum_{i=1}^{m} a_i \right) + c_0 - c_m \qquad \textbf{Equação 5–2}$$

Prova: O resultado final é decorrente do fato de o somatório:

$$\sum_{i=1}^{m} (c_{i-1} - c_i)$$

constituir uma soma telescópica (v. **Apêndice B** do **Volume 1**), de modo que com exceção do primeiro e do último valores, todos os valores cancelam-se mutuamente e, portanto, não entram no cômputo final. ■

Escolhendo uma função de saldo de tal maneira que os valores a_i sejam aproximadamente iguais, é fácil calcular o lado direito da **Equação 5–2** e, portanto, o custo real da sequência de m operações.

5.2.3 Método de Potencial

No método de potencial, associa-se a uma estrutura de dados uma função Φ que representa o estado corrente de energia potencial da estrutura. Cada operação executada sobre essa estrutura irá acrescentar alguma quantia para a função Φ, mas também irá extrair de Φ um valor proporcional ao tempo realmente gasto na operação. Usam-se $\Phi_0 \geq 0$ como valor inicial de Φ (i.e., antes da execução de qualquer operação) e Φ_i como valor da função Φ depois da execução da i-ésima operação. O objetivo aqui é usar a alteração de potencial causada pela i-ésima operação, representada por $\Phi_i - \Phi_{i-1}$, para calcular o custo amortizado necessário para realizar essa operação.

Mais formalmente, define-se o estado de uma estrutura de dados logo após a execução de uma operação i como e_i. A função potencial $\Phi(e_i)$, escrita sucintamente como Φ_i, mapeia e_i num valor real. Denotando-se o custo real da operação i como r_i, o custo amortizado da operação i é igual ao custo real mais a alteração de potencial decorrente da execução da operação:

$$a_i = r_i + \Phi(e_i) - \Phi(e_{i-1}) \qquad \textbf{Equação 5–3}$$

em que r_i é o custo real da operação e e_{i-1} e e_i são os estados da estrutura de dados antes e depois da execução de uma operação, respectivamente.

A **Equação 5–3** pode ser escrita de modo mais sucinto como:

$$a_i = r_i + \Phi_i - \Phi_{i-1} \qquad \textbf{Equação 5–4}$$

Idealmente, a função Φ deve ser definida de modo que o custo amortizado de cada operação seja pequeno. Além disso, a alteração de potencial deve ser positiva para operações de baixo custo e negativa para operações de alto custo.

O custo total amortizado A de execução de n operações numa estrutura de dados é definido como:

$$A = \sum_{i=1}^{n} a_i \qquad \Rightarrow$$

$$A = \sum_{i=1}^{n} (r_i + \Phi_i - \Phi_{i-1}) \qquad \Rightarrow$$

$$A = \sum_{i=1}^{n} r_i + \sum_{i=1}^{n} (\Phi_i - \Phi_{i-1}) \Rightarrow$$

$$A = R + \sum_{i=1}^{n} (\Phi_i - \Phi_{i-1})$$

Como o segundo termo dessa última equação forma uma soma telescópica (v. **Lema 5.1**), obtém-se o seguinte dessa última expressão:

$A = R + \Phi_n - \Phi_0$ **Equação 5–5**

Em palavras, o custo amortizado total é igual ao custo real total mais a alteração de potencial ocorrida durante a execução da sequência inteira de operações. Assim, para garantir que o custo amortizado total seja um limite superior sobre o custo real total (i.e., $A \geq R$) para uma sequência de qualquer tamanho, deve-se assegurar que $\Phi(e_n) \geq \Phi(e_0)$ para todo n. Tipicamente, o valor de $\Phi(e_0)$ é definido como 0 e a função potencial é definida de modo que ela seja sempre não negativa.

O sucesso de uma análise amortizada usando o método de potencial depende de uma boa escolha para a função potencial. Essa função deve ser definida de modo que uma operação cujo custo real seja menor do que seu custo amortizado resulte num aumento de potencial e uma operação cujo custo real seja maior do que seu custo amortizado resulte numa redução de potencial. Assim ela economiza energia suficiente para ser usada quando for necessário, mas não pode economizar energia demais que faça com que o custo amortizado da operação corrente seja alto demais. Infelizmente, a escolha de uma boa função potencial nem sempre é trivial.

5.2.4 Comparação de Métodos de Análise Amortizada

O método contábil cobra por cada operação uma certa quantia de acordo com a natureza da operação, com enfoque em pagamentos prévios que compensem os custos de futuras operações onerosas. Por sua vez, o método de potencial é baseado no efeito de uma operação sobre uma estrutura de dados. Contudo os dois métodos efetuam cobranças (monetária num caso e energética no outro) que compensem um eventual custo adicional para realização de operações futuras. O método contábil *armazena* créditos numa estrutura de dados, enquanto o método de potencial considera propriedades da estrutura de dados para calcular seu *potencial*.

O método contábil e o método de potencial são equivalentes em termos de aplicabilidade a determinados problemas e os custos amortizados que eles proveem. Todavia um deles pode ser mais fácil de aplicar do que o outro num caso específico de análise.

O método de agregado é o mais simples de todos, mas sua aplicabilidade é limitada. Por exemplo, esse método é difícil de aplicar na análise amortizada de árvores afuniladas (v. **Seção 5.4**).

A **Tabela 5–2** apresenta as principais características dos métodos de análise amortizada discutidos aqui.

Método	Características
Agregado	Não faz distinção entre operações com respeito ao custo de cada uma delas
Contábil	Pode atribuir custos (***monetários***) distintos a operações distintas de uma sequência de operações
Potencial	Atribui um valor (***energia potencial***) para cada estado de uma estrutura de dados

Tabela 5–2: Principais Características dos Métodos de Análise Amortizada

5.2.5 Exemplos de Análise Amortizada

Para facilitar o entendimento das técnicas de análise amortizada, será apresentada como exemplo uma estrutura de dados fictícia, que será aqui denominada **tabela de destruição**. Essa tabela é implementada como uma lista simplesmente encadeada e suporta apenas duas operações:

1. Acréscimo, que acrescenta um elemento ao início da tabela.
2. Destruição, que libera o espaço que a tabela ocupa em memória.

O tipo de nó e o tipo de ponteiro para nó da lista são definidos como:

```
typedef struct rotNoLSE {
          int              conteudo;
          struct rotNoLSE *proximo;
       } tNoListaSE, *tListaSE;
```

A operação de acréscimo é simples de implementar, como mostra a função `InsereListaSE()` abaixo.

```
void InsereListaSE(tListaSE *lista, int conteudo)
{
   tNoListaSE *ptrNovoNo; /* Apontará para o novo nó alocado */

   ASSEGURA(ptrNovoNo = malloc(sizeof(tNoListaSE)), /* Tenta alocar */
            "Nao foi possivel alocar no'");  /* um novo nó   */

   ptrNovoNo->conteudo = conteudo; /* Preenche conteúdo do nó */

      /* Faz o novo nó apontar para o primeiro nó da lista */
   ptrNovoNo->proximo = *lista;

   *lista = ptrNovoNo; /* Torna o novo nó o primeiro da lista */
}
```

A operação de destruição também é relativamente trivial, como mostra a função `DestroiListaSE()` apresentada a seguir.

```
void DestroiListaSE(tListaSE *lista)
{
   tListaSE p; /* Aponta para o próximo nó a ser liberado */

   if (!*lista)
      return; /* Lista vazia não precisa ser destruída */

   p = *lista; /* Faz p apontar para o início da lista */

   do { /* Visita cada nó da lista liberando-o */
      *lista = (*lista)->proximo;

      free(p); /* Libera o espaço do nó corrente */

      p = *lista; /* Faz p apontar para o próximo nó */
   } while (p);
}
```

Usando análise assintótica tradicional, claramente, a função `InsereListaSE()` tem custo temporal $\theta(1)$, enquanto a função `DestroiListaSE()` tem custo temporal $\theta(n)$, visto que todos os elementos da tabela devem ser acessados para que o espaço ocupado pela tabela seja liberado. O que será mostrado a seguir é que o custo temporal da operação de destruição implementada pela função `DestroiListaSE()` tem custo amortizado $\theta(1)$.

Usando o Método de Agregado

Considere uma sequência de n operações numa tabela de destruição inicialmente vazia. Se a análise tradicional de pior caso for considerada, o custo temporal dessa sequência de operações é $\theta(n^2)$, visto que o pior caso de

uma única operação de destruição na sequência é $\theta(n)$ e pode haver $\theta(n)$ operações de destruição nessa sequência. Embora essa análise seja correta, ela é um tanto exagerada, pois uma análise que leva em conta as interações entre essas operações mostra que o custo temporal da sequência inteira é realmente $\theta(n)$, como mostra o **Teorema 5.1**.

Teorema 5.1: Uma sequência de n operações numa tabela de destruição inicialmente vazia tem custo temporal $\theta(n)$.

Prova: Considere uma sequência qualquer de n operações de acréscimo e destruição, representadas por chamadas das funções `InsereListaSE()` e `DestroiListaSE()`, numa tabela de destruição inicialmente vazia. O número máximo de execuções do corpo do laço **do-while** da função `DestroiListaSE()` é igual ao número de vezes que a função `InsereListaSE()` foi chamada anteriormente nessa sequência. Portanto o número de execuções desse laço em toda a sequência é menor do que n. Logo o custo temporal de toda a sequência é menor do que $2n$, de modo que esse custo é $\theta(n)$. ■

Como o custo temporal amortizado de uma operação que faz parte de uma sequência de operações é o custo de pior caso da sequência de operações dividido pelo número de operações, pelo **Teorema 5.1**, pode-se dizer que o custo temporal amortizado de cada operação numa tabela de destruição é $\theta(1)$. Note que o custo temporal real de uma operação específica pode ser muito maior do que seu custo temporal amortizado [p. ex., uma determinada operação de destruição pode ter custo temporal $\theta(n)$].

Usando o Método Contábil

Agora será apresentada uma prova alternativa do **Teorema 5.1** obtida por meio do método contábil. Suponha que uma moeda virtual (*V$ 1*) seja suficiente para pagar pela execução de uma operação de acréscimo e pelo acesso a um elemento efetuado por uma operação de destruição. Serão cobradas *V$ 2* para cada uma dessas operações. Isso significa cobrar menos por uma operação de destruição e cobrar *V$ 1* a mais por cada operação de acréscimo. A moeda virtual lucrada numa operação de acréscimo será *armazenada* no elemento acrescentado por essa operação, como mostra a **Figura 5–1**. Quando uma operação de destruição é executada, a moeda virtual armazenada em cada elemento da tabela é usada para pagar pelo tempo gasto acessando-o. Portanto tem-se um esquema de amortização válido, no qual cobram-se *V$* 2 por cada operação, de modo que todo o tempo de processamento é quitado. Esse esquema simples de amortização implica no resultado do **Teorema 5.1**.

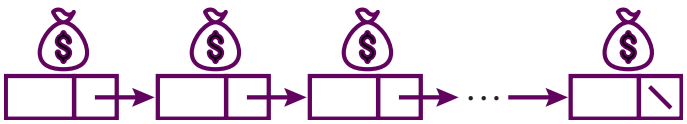


Figura 5–1: Moedas Virtuais Armazenadas numa Estrutura de Dados

Note que o pior caso para o custo temporal ocorre quando uma sequência de operações de acréscimo é seguida por uma única operação de destruição. Em outros casos, ao final da sequência de operações, podem-se lucrar algumas moedas virtuais que não foram gastas, que são aquelas obtidas nas operações de inserção e armazenadas nos elementos que ainda fazem parte da tabela. Por exemplo, se forem efetuadas três acréscimos, seguidas por uma destruição e mais dois acréscimos, nessa ordem, ao final dessa sequência haverá na tabela dois elementos, cada um dos quais com *V$ 1*.

Usando o Método de Potencial

A função potencial Φ a ser usada na análise amortizada de tabela de destruição por meio do método de potencial é escolhida como o número de elementos dessa tabela. O objetivo agora será mostrar que o custo amortizado para qualquer operação é *2*. Isto é, $a_i = 2$, para $1 \leq i \leq n$. Para tal, considere as duas operações possíveis na i-ésima operação:

- **Inserção**. Inserir um nó na tabela acrescenta *1* à função Φ e o custo real r_i necessário é *1* unidade de tempo. Assim, nesse caso, tem-se:
  $a_i = r_i + \Phi_i - \Phi_{i-1} \Rightarrow a_i = 1 + 1 = 2$
- **Destruição**. Remover todos os m elementos de uma tabela de destruição requer, no máximo, $m + 2$ unidades de tempo, sendo m unidades de tempo para executar o laço **do-while** mais, no máximo, *2* unidades de tempo para as demais instruções da função `DestroiListaSE()`. Mas essa operação também reduz o potencial Φ da estrutura de m para *0*, visto que, após essa operação a tabela fica sem nenhum elemento. Portanto, nesse caso, tem-se que:
  $a_i = r_i + \Phi_i - \Phi_{i-1} \Rightarrow a_i = m + 2 + 0 - m = 2$

Portanto o custo temporal amortizado de qualquer operação numa tabela de destruição é $\theta(1)$. Além disso, como $\Phi_i \geq \Phi_0$ para qualquer $i \geq 1$, o custo temporal amortizado para execução de n operações numa tabela de destruição inicialmente vazia é $\theta(n)$.

5.3 Análise Amortizada de Array Dinâmico

Considere uma tabela de busca indexada implementada por meio de um array dinâmico cujo tamanho dobra sempre que mais espaço se faz necessário. Uma tabela como essa foi implementada na **Seção 3.3.2** e é aconselhável que o leitor releia essa seção antes de prosseguir para que possa ficar bem inteirado sobre o problema que será discutido aqui. A **Figura 5–2** ilustra a evolução de uma tabela indexada dinâmica desde seu estado inicial até o instante em que ela contém cinco elementos.

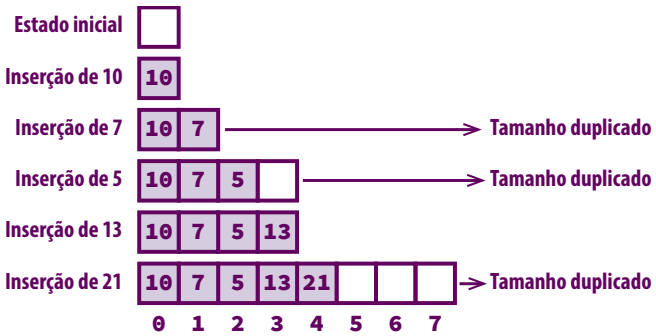


Figura 5–2: Tabela de Busca Indexada Dinâmica

Se o espaço alocado inicialmente para o referido array fosse suficiente para conter quatro elementos da tabela, o custo temporal para inserção dos primeiros quatro elementos dessa tabela seria constante. Contudo a inserção de um quinto elemento nessa tabela levaria mais tempo, visto que o array teria de ser redimensionado antes do acréscimo do novo elemento. As próximas quatro operações de inserção também teriam custo temporal constante. Então um acréscimo subsequente iria requerer outra duplicação do tamanho do array e assim por diante.

Em C, o que retarda o redimensionamento de um array dinâmico é o comportamento da função **realloc()** em seu pior caso, que é ilustrado na **Figura 5–3**[2].

Se for usada análise assintótica tradicional, o custo temporal de inserção numa tabela de busca indexada dinâmica (v. **Seção 3.3.2**) é $\theta(n)$ no pior caso, o que faz com que o custo de pior caso de n inserções seja $\theta(n^2)$. De fato, essa análise resulta num limite superior pessimista demais, pois poucas dessas n inserções têm custo temporal $\theta(n)$.

A seguir, serão apresentadas análises amortizadas de arrays dinâmicos usando os três métodos de análise descritos acima.

[2] Uma discussão completa sobre o comportamento da função **realloc()** é apresentada no **Volume 1**.

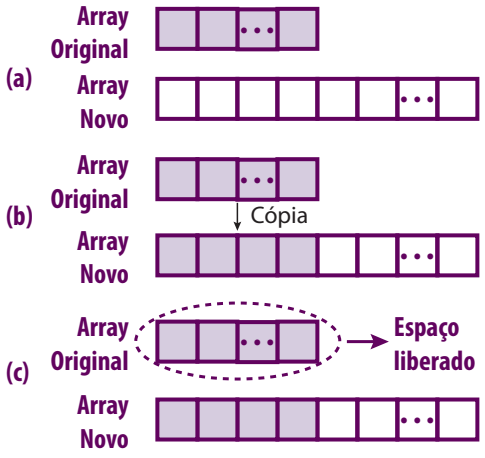


Figura 5–3: Pior Caso da Função `realloc()`

5.3.1 Usando Método de Agregado

O custo c_i da i-ésima inserção numa tabela indexada dinâmica que dobra de tamanho quando necessário é dado por:

$$c_i = \begin{cases} i & \text{se } i-1 \text{ for potência de 2} \\ 1 & \text{em caso contrário} \end{cases}$$

Supondo que o tamanho de uma tabela indexada dinâmica seja t_i, a **Figura 5–4** ilustra as inserções dos 10 primeiros itens numa tabela dessa natureza.

i	1	2	3	4	5	6	7	8	9	10
t_i	1	2	4	4	8	8	8	8	16	16
c_i	1	2	3	1	5	1	1	1	9	1

Figura 5–4: Inserções numa Tabela Indexada Dinâmica 1

O que se pretende é obter a soma dos custos c_i para uma sequência de n inserção e isso é facilitado se for observado que os custos que são diferentes de 1 podem ser reescritos como a soma de 1 com uma potência de 2, como mostra a **Figura 5–5**.

i	1	2	3	4	5	6	7	8	9	10
t_i	1	2	4	4	8	8	8	8	16	16
c_i	1	$1 + 2^0$	$1 + 2^1$	1	$1 + 2^2$	1	1	1	$1 + 2^3$	1

Figura 5–5: Inserções numa Tabela Indexada Dinâmica 2

Como sugere a **Figura 5–5**, a soma dos custos de inserção c_i de uma sequência de n inserções pode ser subdividida em duas partes: (1) a soma das n parcelas iguais a 1 e (2) a soma das parcelas iguais a 2^i, em que i varia entre 0 e $\lfloor \log_2 (n-1) \rfloor$. Ou seja:

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} 1 + \sum_{i=0}^{\lfloor \log(n-1) \rfloor} 2^i$$

Claramente, o primeiro somatório do lado direito resulta em n. Por sua vez, o último somatório é uma soma geométrica cuja razão é 2, de maneira que, usando-se a fórmula de soma de progressão geométrica, obtém-se:

$$\sum_{i=0}^{\lfloor \log(n-1) \rfloor} 2^i = 2^{\lfloor \log(n-1) \rfloor + 1} - 1 = 2 \cdot 2^{\lfloor \log(n-1) \rfloor} - 1 \leq 2 \cdot (n-1) - 1 = 2n - 3$$

Assim o custo temporal total de n inserções é limitado por:

$$\sum_{i=1}^{n} c_i \leq n + 2n - 3 = 3n - 3$$

Portanto esse custo é $\theta(n)$. Como há n operações na sequência, pode-se concluir que o custo temporal amortizado de inserção de um elemento na tabela implementada como array dinâmico descrita acima tem custo temporal $\theta(1)$.

5.3.2 Usando Método Contábil

Suponha que custa *V$ 1* para inserir um elemento e *V$ 1* para copiá-lo quando o array é duplicado. Claramente a cobrança de *V$ 1* por inserção não é suficiente, porque não sobra nada para pagar pela cópia. A cobrança de *V$ 2* por inserção também não é suficiente, mas a cobrança de *V$ 3* parece ser satisfatória, como mostrado na **Figura 5–6**, em que s_i é o saldo depois da i-ésima inserção e p_i é o pagamento efetuado.

i	1	2	3	4	5	6	7	8	9	10
t_i	1	2	4	4	8	8	8	8	16	16
c_i	1	2	3	1	5	1	1	1	9	1
p_i	3	3	3	3	3	3	3	3	3	3
s_i	2	3	3	5	3	5	7	9	3	5

Figura 5–6: Análise Amortizada de Tabela Indexada Dinâmica 1

A prova do **Teorema 5.2** mostra que a cobrança de *V$ 3* por inserção é, de fato, suficiente.

Teorema 5.2: Considere uma tabela implementada por meio de um array dinâmico, como foi descrito acima. O custo temporal total para executar uma sequência de n operações de acréscimo na tabela, inicialmente vazia e usando um array de tamanho inicial $n = 1$, é $\theta(n)$.

Prova: Suponha que *V$ 1* seja suficiente para pagar pela execução de cada operação de inserção na tabela, excluindo o tempo despendido para redimensionar o array. Assuma também que aumentar o tamanho do array de k para $2k$ requer k moedas virtuais (i.e., *V$ k*) para pagar pelo tempo gasto copiando seus elementos. Será cobrado *V$ 3* por cada operação de inserção. Assim cobra-se *V$ 2* a mais por cada operação de inserção que não causa redimensionamento. Imagine que as *2* moedas virtuais lucradas numa inserção dessa natureza sejam armazenadas em cada elemento inserido. Um redimensionamento ocorre quando a tabela tem 2^i elementos, para algum inteiro $i \geq 0$ e o tamanho do array é 2^i. Assim dobrar o tamanho dele irá requer 2^i moedas virtuais, que podem ser encontradas nos elementos armazenados entre as posições 2^{i-1} e $2^i - 1$.

A **Figura 5–7** ilustra essa situação quando o array é duplicado de *8* para *16* elementos. Note que o redimensionamento anterior ocorreu quando o número de elementos se tornou maior do que 2^{i-1} pela primeira vez e assim as moedas virtuais armazenadas nas posições entre 2^{i-1} e $2^i - 1$ não foram previamente gastas. Portanto tem-se um esquema de amortização válido no qual cobram-se *3* moedas virtuais por cada operação que são suficientes para pagar todo o custo de processamento. Ou seja, pode-se pagar pela execução de n operações de acréscimo usando $3n$ moedas virtuais. Portanto o aludido custo temporal é $\theta(n)$. ■

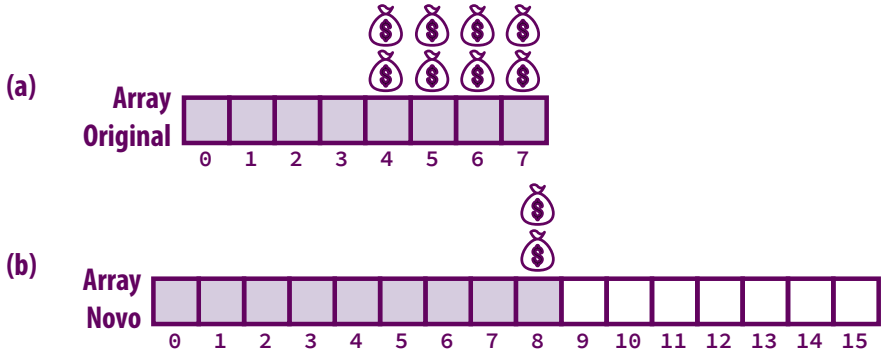


Figura 5–7: Análise Amortizada de Tabela Indexada Dinâmica 2

Usando esse esquema de amortização, sempre que um array é duplicado, essa operação já está paga. Na primeira vez que um elemento é copiado, sua cópia é paga por uma de suas próprias moedas virtuais que foram cobradas quando ele foi inserido. Todas as demais cópias subsequentes desse elemento serão pagas por doações de elementos inseridos posteriormente.

5.3.3 Usando Método de Potencial

Para um array dinâmico cujo redimensionamento é efetuado por duplicação, pode-se usar a seguinte função potencial:

$$\Phi(e_i) = 2n - m$$

em que e é o estado da estrutura, n é o número corrente de elementos e m é a capacidade corrente do array. Se o array for iniciado com capacidade igual a 0 e for alocado um array de tamanho 1 quando o primeiro elemento for acrescentado e, então, dobrar-se o tamanho do array sempre que se precisar de mais espaço, tem-se que: $\Phi(e_0) = 0$. Além disso, como $n \geq m/2$, tem-se que $2n - m \geq 0 \Rightarrow \Phi(e_i) \geq 0, \forall i$.

Para mostrar que o acréscimo de um elemento à tabela tem custo temporal amortizado constante, dois casos devem ser levados em conta:

1. Se $n < m$, o custo temporal amortizado é dado por:

$$\begin{aligned} a_i &= r_i + \Phi_i - \Phi_{i-1} \\ &= 1 + 2\cdot(n+1) - m - (2\cdot n - m) \\ &= 3 \end{aligned}$$

2. Se $n = m$, o custo temporal amortizado é:

$$\begin{aligned} a_i &= r_i + \Phi_i - \Phi_{i-1} \\ &= n + 1 + 2\cdot(n+1) - 2\cdot m - (2\cdot n - m) \\ &= n + 3 - m \\ &= 3 \text{ (já que } n = m) \end{aligned}$$

Em ambos casos, o custo temporal amortizado é $\theta(1)$.

5.3.4 Efeito de Crescimento Geométrico

Qualquer que seja o método utilizado na análise amortizada de array dinâmico, para que o custo temporal amortizado obtido acima seja válido, é crucial que o tamanho do array cresça geometricamente (i.e., dobrando de tamanho). Pode ser tentador aumentar o array por um incremento constante, mas essa abordagem resulta em custo temporal amortizado linear, em vez de custo temporal amortizado constante.

Quando se redimensiona o tamanho do array incrementando-o e são consideradas n operações de inserção, mesmo levando em conta apenas o redimensionamento do array, o custo temporal total será, pelo menos, $1 + 2 + 3 + 4 + \ldots + (n - 1) = n \cdot (n - 1)/2$. Ou seja, o custo amortizado por operação é $(n - 1)/2$ considerando apenas redimensionamento. Por outro lado, quando o tamanho do array é duplicado a cada redimensionamento, em qualquer sequência de n operações, o custo total de redimensionamento é $1 + 2 + 4 + 8 + \ldots + 2^i$ sendo $2^i < n$, que é, no máximo, $2 \cdot n - 1$. Acrescentando-se n como custo adicional para inserção ou remoção, obtém-se que o custo total é menor que $3n$ e, assim, o custo amortizado por operação é menor do que 3.

5.4 Análise Amortizada de Árvores Afuniladas

No **Capítulo 4**, foi visto que o custo temporal de qualquer operação sobre um nó X de uma árvore binária de busca é proporcional ao número de nós encontrados no caminho entre a raiz e X. Além disso, cada operação de busca, inserção ou remoção efetuada numa árvore afunilada requer o afunilamento de um nó, que é efetuado por meio de **passos**.

Esta seção apresentará uma análise amortizada de árvores afuniladas usando o método de potencial[3]. Por questão de simetria, serão considerados apenas os passos de afunilamento zig, zig-zig e zig-zag.

Se cada passo zig numa árvore afunilada contar como uma rotação e cada passo zig-zig ou zig-zag contar como duas rotações, então o custo temporal real de qualquer uma dessas operações será igual a 1 mais o número de rotações.

Para obter o custo temporal amortizado de afunilamento, podem-se enumerar todas as alterações possíveis numa árvore que um único passo de afunilamento pode causar. Então, para cada caso, calcula-se o potencial para a árvore inteira antes e depois dessa operação e o custo real requerido para executá-la. Isso resulta num limite para o custo amortizado de um único passo de afunilamento. Somando-se todos esses custos obtém-se um limite para o custo amortizado de uma operação inteira de afunilamento.

Suponha que $S_i(X)$ seja a subárvore cuja raiz é o nó X no instante i e $|S_i(X)|$ seja o número de nós (**tamanho**) dessa subárvore nesse instante. Define-se o **posto** do nó X como:

$$R_i(X) = \log |S_i(X)|$$

De acordo com essa definição, para uma árvore contendo n nós e cuja raiz seja A, o posto dessa raiz é simplesmente $R(A) = \log n$.

Supondo que $R_i(X)$ indica o posto do nó X de uma árvore cuja raiz é A no instante i, a função potencial Φ para essa árvore nesse instante é definida como:

$$\phi_i(A) = \sum_{X \in A} R_i(X) \qquad \textbf{Equação 5–6}$$

Usar a soma de postos como função potencial é similar a usar a soma de alturas para a mesma finalidade, mas existe uma importante diferença: apesar de uma rotação poder alterar as alturas de muitos nós, no máximo três nós podem ter seus postos alterados em consequência de uma rotação (v. **Figura 5–10**, por exemplo). Se X for uma folha, então $|S_i(X)| = 1$ e, assim, $R_i(X) = 0$. Nós próximos das folhas de uma árvore têm postos pequenos, enquanto a raiz tem o maior posto na árvore.

A **Figura 5–8** ilustra os conceitos de tamanho e posto de um nó e de potencial de uma árvore binária. Nessa figura, a árvore que se encontra no centro e aquela mais à direita são árvores resultantes de afunilamentos ocorridos na árvore à esquerda. Note como os valores dos tamanhos, postos e potenciais variam de uma árvore para outra.

[3] Essa análise é baseada no trabalho de Sleator e Tarjan (1985) (v. **Bibliografia**).

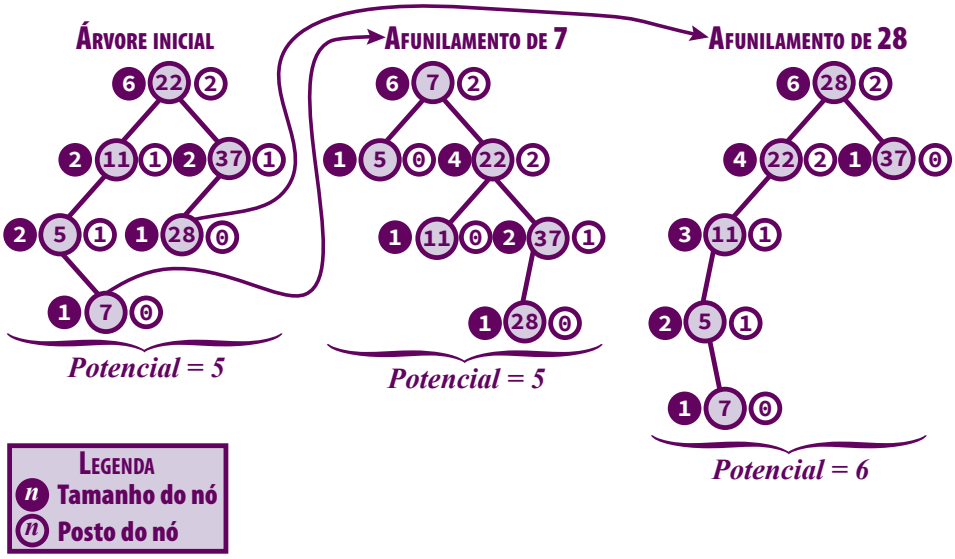


Figura 5–8: Tamanhos e Postos de Nós e Potenciais de Árvores Binárias

O **Lema 5.2**, apresentado a seguir, facilitará a prova do principal resultado desta seção.

Lema 5.2: Se a, b e c são números reais positivos com $a + b \leq c$, então tem-se que: $\log a + \log b \leq 2 \cdot \log c - 2$.

Prova:

$$\left(\sqrt{a} - \sqrt{b}\right)^2 \geq 0 \Rightarrow \sqrt{ab} \leq \frac{a+b}{2} \Rightarrow \sqrt{ab} \leq \frac{c}{2}$$

A última desigualdade acima é decorrente da hipótese: $a + b \leq c$. Elevando-se ao quadrado ambos os lados da última desigualdade e aplicando-se logaritmos na base *2*, obtém-se o resultado desejado. ■

Teorema 5.3: O custo temporal amortizado de afunilamento de um nó X de uma árvore com raiz A é, no máximo, $3(R(A) - R(X)) + 1$, que é $O(\log n)$.

Prova: A função potencial que será usada é a soma dos postos dos nós da árvore A representada na **Equação 5–6**.

Se X for a raiz de A, então não haverá nenhuma rotação e não haverá alteração no potencial. Nesse caso, o custo temporal real para acesso ao nó X é *1*, de maneira que o custo temporal amortizado também é *1* e o teorema é válido.

Suponha agora que há pelo menos uma rotação. Para qualquer passo de afunilamento, sejam $R_i(X)$ e $|S_i(X)|$ o posto e o tamanho (i.e., número de nós) de X antes desse passo sejam $R_f(X)$ e $|S_f(X)|$ o posto e o tamanho de X imediatamente depois do passo de afunilamento. Será mostrado que o custo temporal amortizado requerido para um passo zig é no máximo $3 \cdot (R_f(X) - R_i(X)) + 1$ e que o custo temporal amortizado para um passo zig-zag ou zig-zig é no máximo $3 \cdot (R_f(X) - R_i(X))$.

Caso zig. Para um passo zig, o custo temporal real é *1* (para a única rotação) e a alteração de potencial é $R_f(X) + R_f(P) - R_i(X) - R_i(P)$ (v. **Figura 5–9**). Essa alteração de potencial é fácil de calcular, porque apenas as árvores com raízes X e P mudam de tamanho. Assim, usando a_{zig} para representar o custo temporal amortizado, obtém-se:

$$a_{zig} = 1 + R_f(X) + R_f(P) - R_i(X) - R_i(P)$$

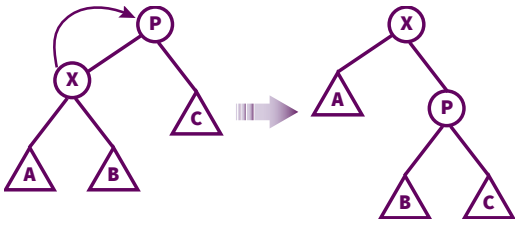


Figura 5–9: Análise Amortizada do Caso Zig de Árvores Afuniladas

Na **Figura 5–9**, observa-se que $|S_i(P)| \geq |S_f(P)|$, de modo que $R_i(P) \geq R_f(P)$ e, assim, tem-se:

$$a_{zig} \leq 1 + R_f(X) - R_i(X)$$

Como $|S_f(X)| \geq |S_i(X)|$, segue-se que $R_f(X) - R_i(X) \geq 0$, de maneira que se pode aumentar o lado direito da última expressão acima, obtendo:

$$a_{zig} \leq 1 + 3 \cdot [R_f(X) - R_i(X)]$$

Caso zig-zag. Para o caso zig-zag, o custo real é 2 (pois há duas rotações) e a mudança de potencial é $R_f(X) + R_f(P) + R_f(Q) - R_i(X) - R_i(P) - R_i(Q)$ (v. **Figura 5–10**), o que resulta no seguinte custo temporal amortizado:

$$a_{zig\text{-}zag} = 2 + R_f(X) + R_f(P) + R_f(Q) - R_i(X) - R_i(P) - R_i(Q)$$

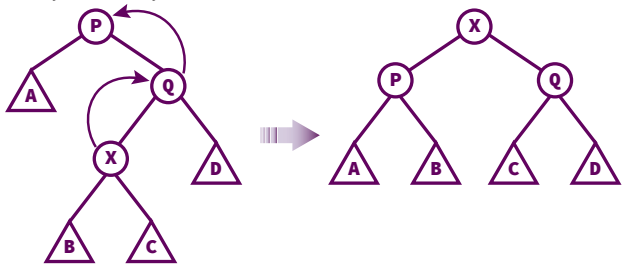


Figura 5–10: Análise Amortizada do Caso Zig-zag de Árvores Afuniladas

Na **Figura 5–10**, vê-se que $|S_f(X)| = |S_i(Q)|$, de maneira que os postos de X e Q são iguais e, assim, obtém-se:

$$a_{zig\text{-}zag} = 2 + R_f(P) + R_f(Q) - R_i(X) - R_i(P) \qquad (\dagger)$$

Nota-se ainda que $|S_i(P)| \geq |S_i(X)|$ (v. **Figura 5–10**), de modo que $R_i(X) \leq R_i(P)$. Fazendo-se essa substituição em **(†)**, obtém-se:

$$a_{zig\text{-}zag} \leq 2 + R_f(P) + R_f(Q) - 2R_i(X) \qquad (\dagger\dagger)$$

Como se vê na **Figura 5–10**, $|S_f(P)| + |S_f(Q)| \leq |S_f(X)|$, de forma que, usando-se esse fato e aplicando-se o **Lema 5.2** em **(††)**, obtém-se:

$$\log |S_f(P)| + \log |S_f(Q)| \leq 2 \log |S_f(X)| - 2$$

Pela definição de posto, essa última expressão pode ser reescrita como:

$$R_f(P) + R_f(Q) \leq 2R_f(X) - 2 \qquad (\dagger\dagger\dagger)$$

Substituindo-se **(†††)** em **(††)**, obtém-se:

$$a_{zig-zag} \leq 2R_f(X) - 2R_i(X) \Rightarrow$$

$$a_{zig-zag} \leq 2(R_f(X) - R_i(X))$$

Finalmente, como $R_f(X) \geq R_i(X)$, obtém-se

$$a_{zig\text{-}zag} \leq 3(R_f(X) - R_i(X))$$

Caso zig-zig. A prova do caso zig-zig (v. **Figura 5–11**) é muito similar ao caso zig-zag. Agora os fatos importantes que devem ser levados em consideração são: $R_f(X) = R_i(Q)$, $R_f(X) \geq R_f(P)$, $R_i(X) \leq R_i(P)$ e $|S_i(X)| + |S_f(Q)| \leq |S_f(X)|$. O complemento da prova desse caso é deixado como exercício para o leitor. Somando-se os custos amortizados de todas as rotações, das quais, no máximo, uma delas pode ser zig (ou zag), vê-se que o custo total amortizado para afunilamento do nó X é no máximo $3 \cdot (R_f(X) - R_i(X)) + 1$, sendo que $R_i(X)$ é o posto de X antes do primeiro passo de afunilamento e $R_f(X)$ é o posto de X depois do último passo de afunilamento. Como o último passo de afunilamento deixa X na raiz da árvore, obtém-se um custo amortizado de $3 \cdot (R(A) - R_i(X)) + 1$, que é $O(\log n)$. ■

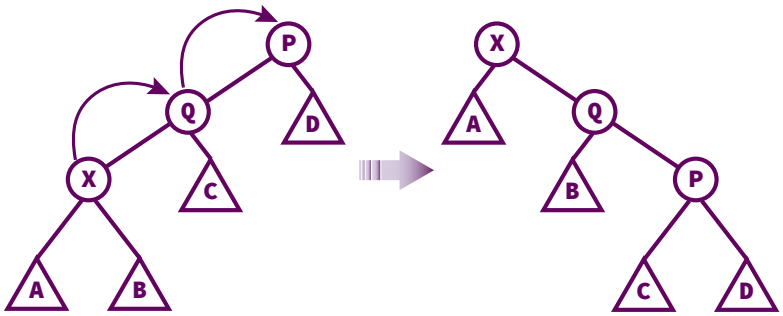


Figura 5–11: Análise Amortizada de Caso Zig-zig de Árvores Afuniladas

A título de ilustração, a **Figura 5–12** mostra os passos que são executados num afunilamento no nó rotulado com *3*. Sejam $R_1(3)$, $R_2(3)$, $R_3(3)$ e $R_4(3)$ os postos do nó *3* em cada uma das quatro árvores. O custo do primeiro passo, que é um zag-zig, é no máximo $3 \cdot (R_2(3) - R_1(3))$. O custo do segundo passo, que é um zig-zig, é $3 \cdot (R_3(3) - R_2(3))$. O último passo é um zig e tem custo que não é maior do que $3 \cdot (R_4(3) - R_3(3)) + 1$. Com o efeito telescópico (v. **Lema 5.1**), o custo total de afunilamento do nó *3* resulta em $3 \cdot (R_4(3) - R_1(3)) + 1$.

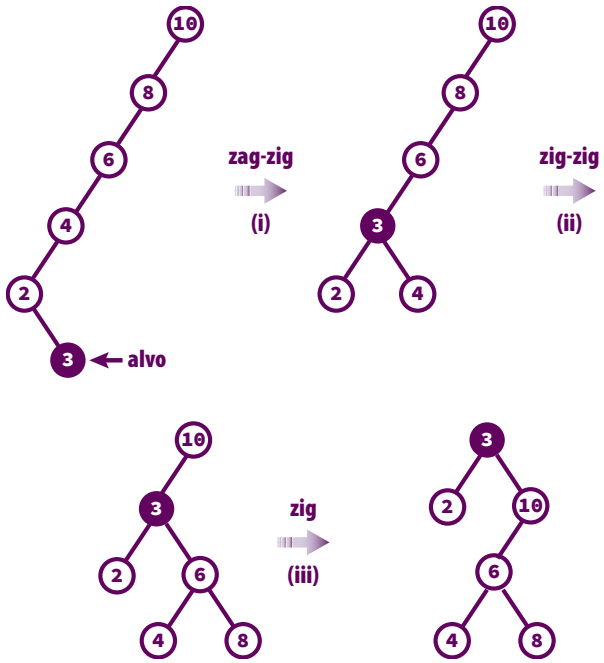


Figura 5–12: Passos de Afunilamento

Corolário 5.1: O custo temporal amortizado de uma operação de busca numa árvore afunilada é $O(\log n)$.

Prova: Como cada operação de busca numa árvore afunilada requer um afunilamento, o custo amortizado de qualquer operação difere por um fator constante do custo amortizado de um afunilamento, que, em consonância com o **Teorema 5.3**, é *O(log n)*. Assim qualquer operação de busca numa árvore afunilada tem custo temporal amortizado *O(log n)*. ■

Corolário 5.2: O custo temporal amortizado de uma operação de inserção numa árvore afunilada é *O(log n)*.

Prova: Como foi visto na **Seção 4.5.3**, no pior caso de inserção, um nó é afunilado e, em seguida um novo nó contendo a chave a ser inserida passa a ser a nova raiz da árvore. Essa segunda operação tem custo *O(1)*, enquanto o afunilamento tem custo amortizado *O(log n)*, de acordo com o **Teorema 5.3**. Portanto o custo amortizado de inserção em árvore afunilada é *O(log n)*. ■

Corolário 5.3: O custo temporal amortizado de uma operação de remoção numa árvore afunilada é *O(log n)*.

Prova: Conforme foi visto na **Seção 4.5.3**, uma remoção consiste em dois afunilamentos seguidos por uma operação que une duas subárvores. Isso aumenta o posto de um nó, mas é limitado por *log n*. Assim o custo de uma remoção é limitado pelo custo de afunilamento, que é *O(log n)*, em virtude do **Teorema 5.3** ■

5.5 Exercícios de Revisão

Fundamentos (Seção 5.1)

1. Descreva os seguintes conceitos:
   (a) Amortização
   (b) Análise amortizada
   (c) Custo amortizado

2. Descreva as principais diferenças entre análise amortizada e análise assintótica.

3. Qual é a semelhança entre análise amortizada e análise assintótica de caso médio.

4. Qual é a principal motivação que norteia a análise amortizada?

5. Quais são os tipos de problemas que podem ser avaliados mediante análise amortizada?

6. Por que análise amortizada não é adequada para análise de algoritmos de ordenação?

7. (a) Cite duas estruturas de dados cujos algoritmos são adequadamente avaliados usando análise amortizada. (b) Cite duas estruturas de dados para as quais análise amortizada não é adequada.

Métodos de Análise Amortizada (Seção 5.2)

8. Descreva os seguintes métodos de análise amortizada:
   (a) Método de agregado
   (b) Método contábil
   (c) Método de potencial

9. (a) O que é moeda virtual no contexto de análise amortizada? (b) Em qual método de análise amortizada moedas virtuais são usadas?

10. Moedas virtuais são armazenadas em estruturas de dados? Explique.

11. O que é soma (ou série) telescópica?

12. (a) O que é função potencial? (b) Quais são as propriedades que uma função potencial deve satisfazer?

13. Explique a analogia entre o método de potencial usando em análise amortizada e energia potencial em Física.

14. (a) Descreva a estrutura de dados tabela de destruição. (b) Para que serve essa estrutura de dados?

Análise Amortizada de Array Dinâmico (Seção 5.3)

15. Por que o tamanho de um array dinâmico deve crescer geometricamente para que a análise amortizada apresentada para esses arrays seja válida?

16. Por que se um array dinâmico for acrescido de um valor fixo quando ele aumenta de tamanho, seu custo amortizado de inserção será $\theta(n)$, e não $\theta(1)$, como no caso mostrado na **Seção 5.3**?

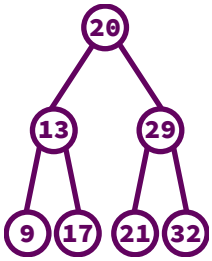
17. Explique por que a seguinte relação é válida:

$$\sum_{i=0}^{\lfloor \log(n-1) \rfloor} 2^i \leq 2n$$

Análise Amortizada de Árvores Afuniladas (Seção 5.4)

18. (a) O que é posto de um nó de uma árvore binária? (b) Qual é a importância desse conceito na análise amortizada de árvores afuniladas?

19. (a) Apresente o posto e o tamanho de cada nó da árvore da figura abaixo. (b) Qual é o potencial dessa árvore?



20. Quais são os custos temporais no pior caso das operações de busca, inserção e remoção em árvores afuniladas?

21. Quais são os custos amortizados das operações de busca, inserção e remoção em árvores afuniladas?

22. (a) Usando a função potencial descrita na **Seção 5.4** para análise amortizada de árvores afuniladas, qual é o potencial máximo e o potencial mínimo de uma árvore afunilada? (b) Quanto o potencial pode diminuir num afunilamento? (c) Quanto o potencial pode aumentar num afunilamento? Apresente suas respostas usando a notação ó.

23. Apresente uma análise amortizada de operações sobre árvores afuniladas usando o método contábil.

24. Considere as estruturas de dados lista com saltos, árvore AVL e árvore afunilada que são usadas em implementações de tabelas de busca. É correto afirmar que todas essas estruturas apresentam custo temporal $\theta(\log n)$ para operações de busca, inserção e remoção?

25. Mostre que o custo amortizado de um afunilamento zig-zig é, no máximo, $3(R_f(X) - R_i(X))$.

26. Mostre que o custo amortizado de um afunilamento zag-zag é, no máximo, $3(R_f(X) - R_i(X))$.

27. Uma função potencial para análise amortizada de árvores afuniladas poderia ser definida como a soma das alturas de todos os nós da árvore. Explique por que essa função não funcionaria.