PCC104 - Projeto e Análise de Algoritmos

Marco Antonio M. Carvalho

Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto

Conteúdo

Fonte

Este material é baseado nos livros

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- S. Halim. *Competitive Programming*. 3rd Edition, 2013.
- Ian Parberry and William Gasarch. *Problems on Algorithms*. Second Edition, 2002.
- Ian Parberry *Lecture Notes on Algorithm Analysis and Complexity Theory*. Fourth Edition, 2001.

Licença

Impacto do Balanceamento

Importância

Como vimos anteriormente, o balanceamento entre os tamanhos dos subproblemas gerados no D&C não é obrigatório, pois é possível gerarmos subproblemas de tamanhos diferentes.

Entretanto, em boa parte dos problemas, devemos levar em consideração tal balanceamento, principalmente por estar diretamente ligado ao esforço necessário para a resolução dos problemas.

O balanceamento se torna especialmente primordial quando utilizamos técnicas de paralelismo.

Não só o D&C se preocupa com o balanceamento de subproblemas, como veremos em breve.

Antes, vejamos um exemplo de impacto na complexidade obtido por meio do balanceamento de subproblemas.

Impacto do Balanceamento

Exemplo

Consideremos o algoritmo de ordenação de n elementos descrito abaixo.

A cada iteração um elemento é inserido em sua posição ideal.

O processo é repetido $n-1$ vezes, pois a cada vez, um subproblema de tamanho $n-1$ é resolvido.

1 **Inserção**(A, n)
  Entrada: Vetor A, inteiro n
2 **para** $i \leftarrow 1$ ***até*** $n-1$ **faça**
3 | Selecione o menor elemento de $A[i..n]$ e o troque de posição com o elemento $A[i]$.
4 **fim**
5 **retorna** A;

Impacto do Balanceamento

Exemplo

O número de comparações realizadas pelo algoritmo anterior pode ser expresso por meio da seguinte recorrência:

- $T(1) = 0$;
- $T(n) = T(n-1) + n - 1$.

Expandindo, temos:

$$\begin{aligned} T(n) &= T(n-1) + n - 1 \\ T(n-1) &= T(n-2) + n - 2 \\ &\vdots \\ T(2) &= T(1) + 1 \\ T(1) &= 0 \end{aligned}$$

O que nos dá $T(n) = 0{+}1{+}2{+}\ldots{+}n-1 = \frac{n(n-1)}{2} = O(n^2)$.

Impacto do Balanceamento

Exemplo

Embora o algoritmo não siga rigorosamente o estereótipo do D&C, pode ser entendido perfeitamente como tal.

Claramente, este algoritmo não é eficiente para valores grandes de n.

Como em outros casos, o balanceamento do tamanho dos subproblemas pode nos trazer ganhos em termos de complexidade.

Analisando superficialmente a recorrência anterior, é possível perceber que a complexidade melhorará com o balanceamento.

Impacto do Balanceamento

Princípio

Dividir o vetor a ser ordenado em dois sucessivamente, até obter n vetores de tamanho unitário.

Gerar um vetor auxiliar, inserindo a cada momento o menor valor dentre dois subproblemas anteriores (já ordenados).

Repete-se este processo, combinando os subproblemas, até que todos os elementos sejam transferidos para o vetor auxiliar, que estará ordenado.

Ao final, os elementos do vetor auxiliar são transferidos para o vetor original.

Impacto do Balanceamento

```
1 BalancedSort(A, i, j)
  Entrada: Vetor A, índices i e j
2 se i < j então
3 |   m ← ⌊(i+j)/2⌋;
4 |   BalancedSort(A, i, m);
5 |   BalancedSort(A, m + 1, j);
6 |   Combinar(A, i, m, j);
7 fim
```

Impacto do Balanceamento

Comentários

O passo de combinação recebe um vetor com duas partes ordenadas $A[1..m]$ e $A[m+1..j]$ e produz um outro vetor ordenado $A[1..j]$.

Como os dois vetores estão ordenados, o passo de combinação requer no máximo $n-1$ combinações.

Novamente, o passo de combinação seleciona repetidamente o menor elemento entre os dois vetores da entrada. Em caso de empate, seleciona qualquer um.

Por simplicidade, consideramos n sendo uma potência de 2.

Impacto do Balanceamento

Análise

As comparações realizadas pelo algoritmo são representadas por:

- $T(1) = \Theta(1)$;
- $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.

Aplicando o Teorema Mestre temos:

- $a = 2$;
- $b = 2$;
- $f(n) = n$.

Logo, $n^{log_b a} = n^{log_2 2} = n$.

Aplicamos o segundo caso do Teorema Mestre e temos que $T(n) = \Theta(n\ log\ n)$.

Claramente, este algoritmo de ordenação é o **Merge Sort** e o passo de combinação é a **intercalação**.

Impacto do Balanceamento

Análise

As comparações realizadas pelo algoritmo são representadas por:

- $T(1) = \Theta(1)$;
- $T(n) = 2T(\frac{n}{2}) + \Theta(n)$.

Aplicando o Teorema Mestre temos:

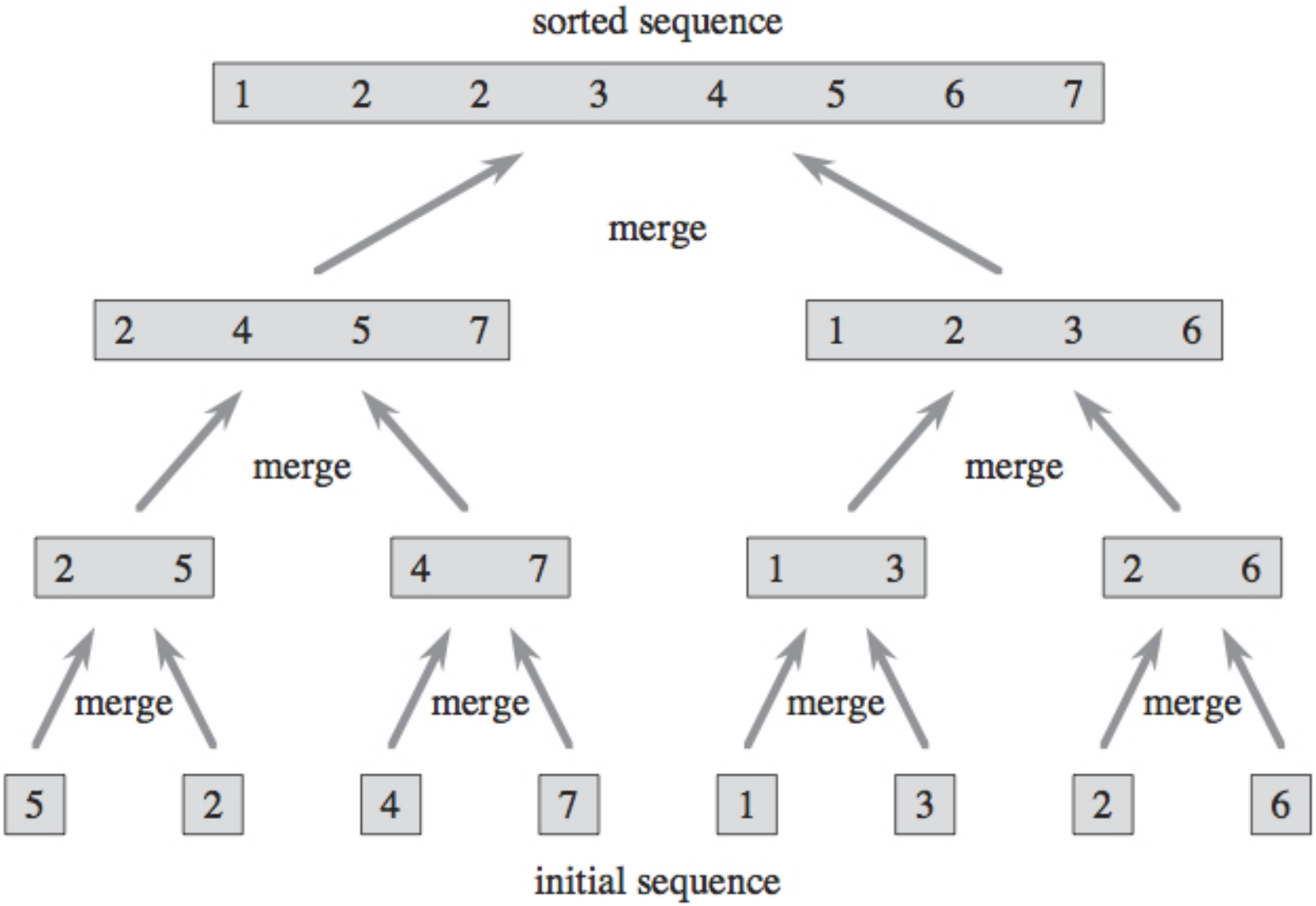
- $a = 2$;
- $b = 2$;
- $f(n) = n$.

Logo, $n^{log_b a} = n^{log_2 2} = n$.

Aplicamos o segundo caso do Teorema Mestre e temos que $T(n) = \Theta(n\ log\ n)$.

Claramente, este algoritmo de ordenação é o **Merge Sort** e o passo de combinação é a **intercalação**.

Impacto do Balanceamento



Uso Tradicional da Busca Binária

Tradicionalmente, utilizamos a busca binária para buscar um elemento em uma sequência estática e ordenada:

- Verificamos se o elemento central da sequência é o que estamos buscando (considerando também a possibilidade de o elemento procurado não existir);
- Caso não seja, verificamos os elementos à esquerda ou à direita, de acordo com os valores do elemento central e do elemento buscado.

À medida em que a busca avança, o espaço de busca é diminuído pela metade, e a complexidade do algoritmo é $O(log\ n)$.

Devido à ausência do passo de combinação, alguns autores[a] classificam a Busca Binária como "Diminuir (pela metade)" e Conquistar.

Curiosamente, esta não é a única maneira de utilizarmos a Busca Binária.

[a] Anany Levitin. Introduction to The Design & Analysis of Algorithms. Addison Wesley, 1st edition, 2002.

Busca Binária Pela Solução

Princípio

A **Busca Binária Pela Solução** é uma estratégia poderosa que consiste em tentarmos determinar a solução de um problema da mesma forma como buscamos um elemento na busca binária tradicional.

Exemplo 1

Dados m livros, enumerados de 1 a m que podem ter um número diferente de páginas $(p_1, p_2, \ldots, p_m)$, desejamos copiá-los todos.

Para tanto, precisamos dividir estes livros entre k escribas $(k \leq m)$. Cada livro só pode ser designado a um escriba, e cada escriba deve ser responsável por uma sequência contígua de livros.

Em outras palavras, há uma sequência contígua e crescente de números $0 = b_0 < b_1 < b_2 < \ldots < b_{k-1} \leq b_k = m$ tal que o i-ésimo escriba é responsável pelos livros cujos números estão entre $b_{i-1} + 1$ e b_i.

O objetivo é minimizar o número máximo de páginas designada a um único escriba.

Busca Binária Pela Solução

Exemplo 1

Há uma solução para este problema utilizando o paradigma de Programação Dinâmica, porém, podemos tentar "adivinhar" a solução utilizando o princípio da Busca Binária!

Suponhamos $m = 9, k = 3$ e $p_1, p_2, \ldots, p_9$ valem 100, 200, 300, 400, 500, 600, 700, 800 e 900, respectivamente.

Chute 1

Vamos chutar que a solução é 1000.

Desta forma, o problema se transforma em: "Se o escriba com mais trabalho for responsável por 1000 páginas, há uma solução viável?". A resposta é "Não".

Podemos designar os livros de maneira gulosa, da seguinte forma: {100, 200, 300, 400} para o escriba 1, {500} para o escriba 2, {600} para o escriba 3 e sobrariam três livros, {700, 800, 900} sem escribas.

Disto, podemos concluir que a solução é maior do que 1000.

Busca Binária Pela Solução

Chute 2

Vamos chutar que a solução é 2000.

Podemos designar os livros de maneira gulosa, da seguinte forma: {100, 200, 300, 400, 500} para o escriba 1, {600, 700} para o escriba 2, {800, 900} para o escriba 3.

Há uma sobra potencial de {500, 700, 300} para os 3 escribas, de maneira que a solução não é maior do que 2000.

Busca Binária Pela Solução

Idéia

A solução para este problema pode ser "buscada binariamente" no intervalo $[i..f]$, em que

- $i = \min(p_i), \forall i \in [1..m]$ (i.e., o número de páginas do menor livro); e
- $f = p_1 + p_2 + \ldots + p_m$ (i.e., todas as páginas).

Solução

A solução ótima é 1700: {100, 200, 300, 400, 500} para o escriba 1, {600, 700} para o escriba 2, {800, 900} – o chute 2.

A complexidade para obtenção desta solução é $O(m\ log\ f)$.

Método da Bisseção

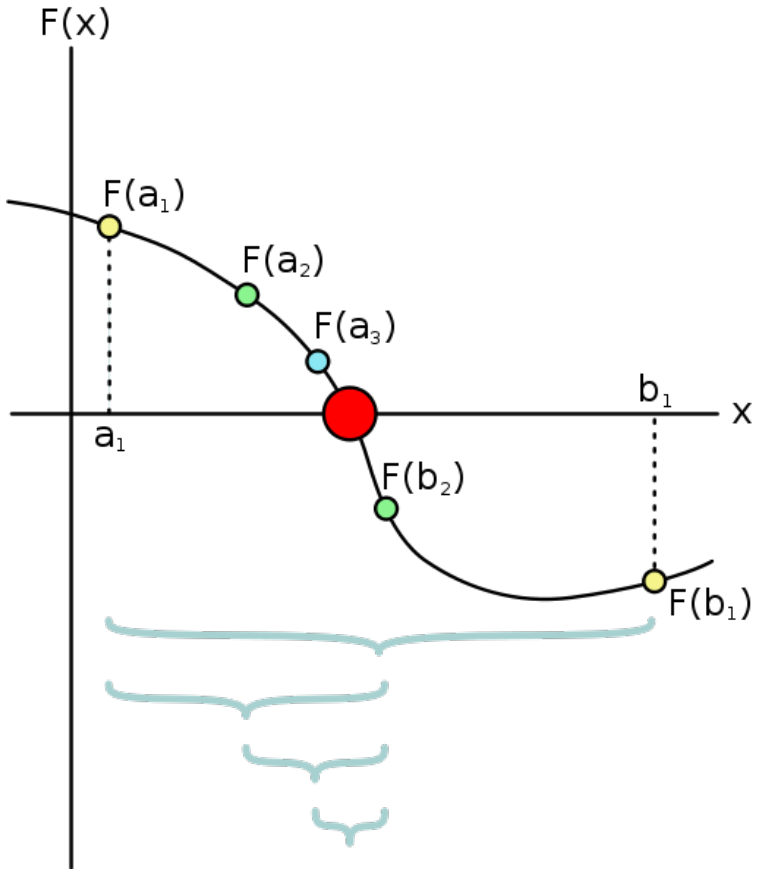
O **Método da Bisseção** é um método numérico simples utilizado para determinarmos as raízes de uma função que eventualmente pode ser difícil de computar matematicamente, tornando-o lento.

Com efeito, o Método de *Newton* ou o Método das Secantes são mais efetivos.

O método repetidamente divide em dois um intervalo $f : [a, b]$ de uma função contínua $f(x)$ e então seleciona um subintervalo, que garantidamente contém a raiz (vide teorema do valor intermediário), para a próxima operação.

Para que o método da bisseção funcione, precisamos nos assegurar que $f(a)$ e $f(b)$ possuam sinais diferentes.

Busca Binária Pela Solução



Método da bisseção aplicado ao intervalo inicial $[a_1, b_1]$.
A primeira iteração gera o ponto b_2, a segunda gera a_2 e a terceira a_3.
O método converge para o ponto em vermelho, que é a raiz da função.

Busca Binária Pela Solução

Exemplo 2

Queremos comprar um carro financiado pagando d dinheiro por m meses.

O valor original do carro é v, e o banco cobra $i\%$ de juros por mês sobre o total devido.

Qual é o valor d que você deve pagar por mês (considerando apenas dois dígitos após a vírgula) para financiar um carro que custa v inicialmente?

Supondo $d = 576, m = 2, v = 1000$ e $i = 10\%$, após um mês sua dívida se torna $1000 \times 1,1 - 576 = 524$. Após 2 meses, temos $524 \times 1,1 - 576 \approx 0$.

Exemplo 2

Porém, se temos apenas $m = 2, v = 1000$ e $i = 10\%$, como determinar $d = 576$?

Em outras palavras, como encontramos a raiz d tal que a função de pagamento da dívida $f(d, 2, 1000, 10) \approx 0$?

Uma maneira é o método da bisseção:

- Selecionamos um intervalo razoável para encontrarmos d, por exemplo $[a \ldots b]$;
- $a = 1$ define que temos que pagar algum valor ($d \geq 1$);
- $b = (1 + i) \times v$ determina o maior valor que podemos pagar e saldar a dívida em um mês. Por exemplo, se $m = 1$, pagaremos $(1 + i) \times v = (1 + 10) \times 1000 = 1100$ após um mês.

Aplicamos o método da bisseção conforme descrito a seguir.

Busca Binária Pela Solução

Exemplo 2

- Se pagarmos $d = (a + b)/2 = (1 + 1100)/2 = 550,5$ por mês, teremos errado por 53,95 a menos após dois meses, então temos que aumentar o pagamento mensal;
- Se pagarmos $d = (550,5 + 1100)/2 = 825,25$ por mês, erramos por 523,05 a mais após dois meses, então temos que diminuir o pagamento;
- Se pagarmos $d = (550,5 + 825,25)/2 = 687,875$ por mês, erramos por 234,5375 a mais após dois meses, então temos que diminuir o pagamento;
- Após $O(log_2((b - a)/\epsilon))$ iterações, em que ϵ é o erro a ser tolerado, temos que...
- Finalmente, se pagarmos 576,190476 por mês, pagaremos corretamente a dívida após dois meses, portanto, $d = 576$ é a resposta.

Observações sobre o exemplo 2

Neste exemplo, seriam necessárias $1099/\epsilon$ tentativas. Usando $\epsilon = 1e-9$, teríamos aproximadamente 40 tentativas.

Usando $\epsilon = 1e-15$, teríamos aproximadamente 60 tentativas.

O método da busca binária pela solução é mais eficiente se comparado com a busca linear de cada valor possível para $d = [1..1100]/\epsilon$.

Exemplo 3

Você está projetando seu próprio veículo para realizar a travessia de um deserto e, preocupado com a autonomia do veículo, você enumera os seguintes eventos relacionados à travessia:

- Consumo de combustível n: seu veículo necessita de n litros de combustível para cruzar 100 km. O consumo varia no intervalo [1..30], de acordo com as características do trecho;
- Vazamento: o tanque de combustível pode sofrer alguma avaria e vazar a uma taxa de 1 litro por km;
- Abastecimento: você atinge um ponto de abastecimento e pode encher o tanque;
- Mecânica: você atinge um ponto de apoio e pode consertar seu tanque, caso esteja vazando.

Busca Binária Pela Solução

Exemplo 3

A organização fornece uma projeção dos eventos que ocorrerão durante a travessia (no máximo 50) e a quilometragem de cada um em relação ao início da travessia.

O objetivo é projetar o menor tanque de combustíveis que possua capacidade de fazer seu veículo atravessar o deserto.

Se soubéssemos de antemão qual é a capacidade do tanque, este seria um problema de simulação – simplesmente simularíamos cada evento e verificaríamos se seria possível atravessar o deserto.

Busca Binária Pela Solução

Análise

O intervalo de possíveis respostas é [0,000..10.000,000], porém, há $10M$ possibilidades dados os 3 dígitos de precisão, o que levaria a busca completa a ter um alto tempo de execução.

Este problema possui uma característica que nos interessa. Suponhamos que a solução é X:

- Qualquer valor entre [0,000 .. X-0,001] não é suficiente para atravessar o deserto;
- Qualquer valor entre [X..10.000,000] é suficiente para atravessar o deserto, eventualmente com sobra de combustível.

Esta propriedade nos permite "buscar binariamente" pela solução! Basta simular os eventos para os valores da capacidade do tanque de combustível dados pela busca binária.

Busca Binária Pela Solução

Exemplos de entradas

0 Fuel consumption 5
100 Fuel consumption 30
200 Goal
[resposta: 35 litros]
0 Fuel consumption 20
10 Leak
25 Leak
25 Fuel consumption 30
50 Gas station
70 Mechanic
100 Leak
120 Goal
[resposta: 81 litros]

Dúvidas?