Projeto e Análise de Algoritmos

Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

17 de maio de 2016

Projeto de Algoritmos por Divisão e Conquista

Uso da Indução em Algoritmos

- Algoritmos projetados pela técnica da Indução são naturalmente recursivos.
- Algoritmos recursivos construídos pela Indução buscam resolver o problema chamando o próprio algoritmo para resolver um problema menor.
- O conjunto caracteriza-se em três partes em cada nível de recursão: Dividir, Conquistar e Combinar.

Tecnica da Divisão e Conquista

1. **Divisão:** O problema é dividido em subproblemas, semelhantes ao problema original porém tendo como entrada instâncias de tamanho menor.
2. **Conquista:** Cada problema menor é resolvido recursivamente, a menos que ele seja suficientemente pequeno para ser resolvido diretamente.
3. **Combinação:** As soluções dos subproblemas são combinadas para obter uma solução do problema maior.

Exemplo da Técnica

Já usamos esta técnica no algoritmo MERGE_SORT

- **Divisão:** Dividimos o conjunto de entrada em dois conjuntos de mesmo tamanho (vamos supor que o tamanho da entrada é uma potência de 2).
- **Conquista:** Aplicamos recursivamente, exceto se o tamanho do conjunto for 1, neste caso o conjunto já está ordenado.
- **Combinação:** As soluções de cada um dos conjuntos, seqüências ordenadas, são intercaladas para obter um conjunto único ordenado.

Aplicando Divisão e Conquista para ordenação

- Divisão: Divida o vetor $A[p..r]$ em dois subvetores $A[p..q-1]$ e $A[q+1..r]$, tais que:

	p	q	r
A	$\leqslant\ x$	x	$>\ x$

$$A[p..q-1] \leqslant A[q] < A[q+1,p]$$

- Conquista: Ordene os dois subvetores recursivamente utilizando o QuickSort. Se somente houver um ou dois elementos, ordene diretamente.
- Combinação: Nada a fazer, o vetor está ordenado.

Divisão dos Vetores: Partição

Problema:

Rearranjar um dado vetor $A[p..r]$ e devolver um índice q, $p \leqslant q \leqslant r$ tais que:

$$A[p..q-1] \leqslant A[q] < A[q+1..r]$$

Exemplo:

Entrada:

$$A \quad \begin{array}{l} \begin{array}{cccccccccc} p & & & & & & & & & r \end{array} \\ \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 5 & 3 & 7 & 5 & 9 & 4 & 8 & 2 & 7 & 6 \\ \hline \end{array} \end{array}$$

Saída:

$$A \quad \begin{array}{l} \begin{array}{cccccccccc} p & & & & & q & & & & r \end{array} \\ \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline 5 & 3 & 5 & 4 & 2 & 6 & 8 & 9 & 7 & 7 \\ \hline \end{array} \end{array}$$

Particionando

	p									r
A	5	3	7	5	9	4	8	2	7	6
i	j									x
A	5	3	7	5	9	4	8	2	7	6
	i	j								x
A	5	3	7	5	9	4	8	2	7	6
		i	j							x
A	5	3	7	5	9	4	8	2	7	6
		i		j						x
A	5	3	5	7	9	4	8	2	7	6
			i		j					x
A	5	3	5	7	9	4	8	2	7	6
			i			j				x

Particionando

A	5	3	5	7	9	4	8	2	7	6
			i			j				x
A	5	3	5	4	9	7	8	2	7	6
				i			j			x
A	5	3	5	4	9	7	8	2	7	6
				i				j		x
A	5	3	5	4	2	7	8	9	7	6
					i				j	x
A	5	3	5	4	2	7	8	9	7	6
					i					jx
A	5	3	5	4	2	6	8	9	7	7
	p					q				r

Algoritmo Particione

Algoritmo PARTICIONE(A,p,r)

$x \leftarrow A[r]$ ▷ x é o "pivô"

$i \leftarrow p - 1$

para $j \leftarrow p$ **até** $r - 1$ **faça**

  se $A[j] \leqslant x$ **então**

    $i \leftarrow i + 1$

    $A[i] \leftrightarrow A[j]$

$A[i+1] \leftrightarrow A[r]$

Retorna $i + 1$

Invariantes:

No começo de cada iteração da linha três vale que:

(1) $A[p..i] \leqslant x$ (2) $A[i+1..j-1] > x$ (3) $A[r] = x$

O QuickSort

Algoritmo QUICK_SORT(A, p, r)
  se $p < r$ **então**
    $q \leftarrow PARTICIONE(A, p, r)$
    $QUICK\_SORT(A, p, q-1)$
    $QUICK\_SORT(A, q+1, r)$

- Vamos calcular o tempo para o algoritmo.
- É simples de observar que para o PARTICIONE, $T(n) \in \Theta(n)$
- Mas como fica para o QUICK_SORT? Não sabemos onde será a partição.

Calculando o tempo de execução do QuickSort

#	QUICK_SORT(A, p, r)	Tempo
2.	**se** $p < r$	$\Theta(1)$
3.	$q \leftarrow PARTICIONE(A, p, r)$	$\Theta(n)$
4.	$QUICK\_SORT(A, p, q-1)$	$T(k)$
5.	$QUICK\_SORT(A, q+1, r)$	$T(n-k-1)$

$$T(n) = T(k) + T(n-k-1) + \Theta(n+1)$$

$$0 \leqslant k \equiv q - p \leqslant (n-1)$$

$$T(0) \in \Theta(1)$$

$$T(1) \in \Theta(1)$$

Complexidade do QuickSort no “PIOR” Caso

- Vamos supor que o conjunto já está ordenado.
- Vamos encontrar $q = n$ em todas as partições que realizamos.
- Ficamos com:

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

- Calculando a recorrência (exercício) encontramos que: $T(n) \in \Theta(n^2)$
- Ou seja, no “PIOR” caso, o QuickSort é $O(n^2)$ e $\Omega(n^2)$

Complexidade do QuickSort no "MELHOR" Caso

- Vamos supor a distribuição que exista de elementos permita que sempre o "pivô" fique em uma posição central.
- Vamos ficar com duas partições de mesmo tamanho.
- Ficamos com:

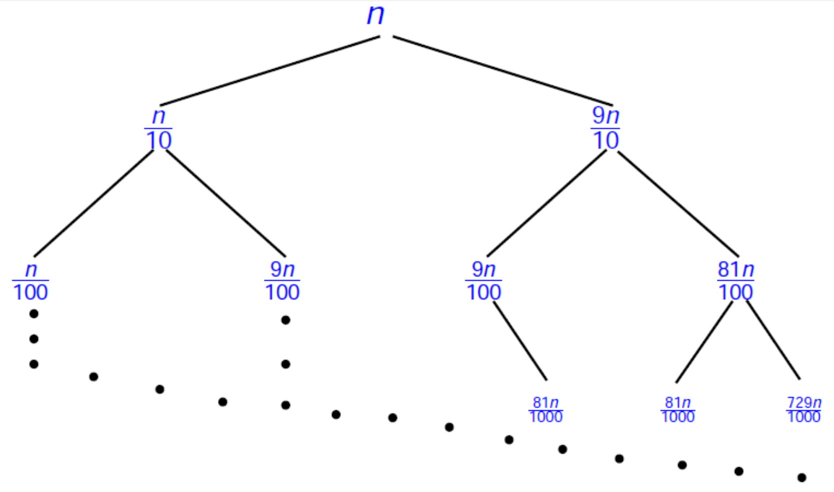
$$T(n) = T\left(\frac{n-1}{2}\right) + T\left(\frac{n-1}{2}\right) + \Theta(n)$$

- Calculando a recorrência (exercício) encontramos que: $T(n) \in \Theta(n \log n)$
- Ou seja, no "MELHOR" caso, o QuickSort é $O(n \log n)$ e $\Omega(n \log n)$

Complexidade do QuickSort no "Caso Médio"

- O cálculo do caso médio para o QuickSort é um pouco extenso, e pode ser visto no CLRS.
- Podemos passar uma idéia que justifique o resultado, embora é apenas uma aproximação grosseira.
- Sempre que há a partição, uma fração do conjunto fica em uma partição e o restante em outra.
- Vamos supor que sempre ocorra a divisão em $\frac{1}{10}$ e $\frac{9}{10}$.
- Construindo a árvore de recorrência vamos perceber que o ramo mais longo tem comprimento $\leqslant \log_{10/9} n$.
- Em cada nível, o custo é $\leqslant n$.
- Mesmo assim, $T(n)$ é $\Theta(n \log n)$.
- O mesmo ocorreria em uma fração $\frac{1}{100}$ e $\frac{99}{100}$.

Árvore de Recorrência para o QuickSort



Problema do cálculo do balanceamento de uma Árvore Binária

Problema

É possível provar por indução que eu consigo calcular se uma árvore está balanceada ou não

- Uma árvore binária é dita desbalanceada se o *Fator de Desbalanceamento* - *fb* em módulo for maior que 1.
- O fator de desbalanceamento é dado pela diferença da altura medida pelo filho esquerdo e pelo filho direito.
- A árvore AVL é um exemplo de árvore binária em que o *fb* se restringe a valores -1, 0 e 1.

Aplicando Divisão e Conquista no Problema

Podemos reduzir o problema a: *Calcular os fatores de desabalanceamento para todos os nós de uma árvore binária*

- Divisão: Considero cada filho do nó *raiz* da árvore um problema menor.
- Conquista: Aplico recursivamente o algoritmo em cada filho, exceto se o filho for uma Árvore Vazia: $fb = 0$.
- Combinação: Com o resultado dos filhos eu verifico se a árvore original baseada no nó *raiz* é balanceada.

Hipótese da Indução

Eu sei calcular o fator de desbalanceamento de todos os nós para uma árvore com menos do que n nós.

Indução Forte

Trabalhando a Combinação - Passo da Indução

- Eu tenho o fator de desbalanceamento de cada filho, obtido na Conquista.
- Preciso calcular o fator de desbalanceamento do raiz.
- **Problema!** Eu só consigo calcular o fator de desbalanceamento se eu tiver a altura dos filhos.
- Precisamos de uma nova hipótese de indução, uma hipótese mais forte:

Hipótese de Indução

Eu sei calcular o fator de desbalanceamento de todos os nós e a altura para uma árvore com menos do que n nós.

Trabalhando a Combinação - Passo da Indução

- O fator de desbalanceamento no caso base (Árvore Vazia) é 0, a altura é -1.
- Pela Conquista, eu obtive: fb_d, h_d, que representam o fator de desbalanceamento do filho direito e a altura do filho direito, bem como fb_e e h_e.
- Agora eu sei calcular para a árvore original: $fb = h_e - h_d$, lembrando que se algum filho tiver um fator de desbalanceamento maior, a árvore está desbalanceada, logo fb $=$ max (abs(fb), f_d, f_e) e $h = max(h_e, h_d) + 1$.

Algoritmo para o cálculo do fator de desbalanceamento

Algoritmo FATOR_ALTURA(A)

se $\acute{E}Vazia(A)$ **então**

Retorna $(0, -1)$

senão

$(fb_e, h_e) \leftarrow FATOR\_ALTURA(FilhoEsq(A))$

$(fb_d, h_d) \leftarrow FATOR\_ALTURA(FilhoDir(A))$

$fb \leftarrow h_e - h_d$

$fb = max(abs(fb), fb_e, fb_d)$

$h \leftarrow max(h_e, h_d) + 1$

Retorna (fb, h)

Considerações finais

- Divisão e Conquista representa um projeto de algoritmo por indução.
- Ao se aplicar a indução, o conjunto menor deve possuir as mesmas propriedades do conjunto maior.
- O passo da indução deve ser aplicável na base, caso contrário a base não representa a base da indução.
- Por vezes, a Hipótese da Indução não é suficiente para se resolver o passo, é necessário fortalecer a hipótese da indução.

Atividades baseadas no CLRS

- Ler o capítulo 2.3
- exercícios: 2.3-1, 2.3-3, 2.3-4
- Ler capítulo 7.1 e 7.2 (QuickSort)
- exercícios: 7.1-1, 7.1-2, 7.2-2