CI1056: Algoritmos e Estruturas de Dados II

Prof. Dr. Marcos Castilho

Departamento de Informática/UFPR

4 de dezembro de 2020

Resumo

Análise da complexidade do *build_max_heap*

Objetivos da aula

- Fazer a análise do algoritmo *build_max_heap*
- Contaremos o número de comparações feitas entre elementos do vetor

Observações gerais

- Esta análise não é simples, mostraremos para fins de completude e também porque o resultado é surpreendente!
- Não será cobrado nas avaliações pois é a única análise deste tipo nesta disciplina
- A análise será portanto menos precisa do que poderia
- Aguardem a disciplina de Análise de Algoritmos para uma melhor análise

O algoritmo *build_max_heap*

```
build_max_heap (v, n)
    para i de n/2 ate 1, regressivamente faca
        max_heapify (v, i)
```

Análise da complexidade de pior caso

- O laço executa exatamente $\lfloor \frac{n}{2} \rfloor$ vezes
- A cada iteração chama o `max_heapify` a um custo *de pior caso* igual a altura da respectiva subárvore
- Na primeira iteração, `max_heapify` será chamado em uma subárvore de altura zero
- Na segunda, a altura será 1, depois 2, ...
- Na última, quando está no índice 1 (raiz), a altura é $\lfloor log_2 n + 1 \rfloor$
- No pior caso, é a soma das alturas de cada subárvore (limitantes superiores), somados $\lfloor \frac{n}{2} \rfloor$

Análise superficial

- Em uma análise incauta pode-se pensar que o custo é $\frac{n}{2} log_2(n)$
- Mas não é!
- Observe que em boa parte da execução do algoritmo as alturas dos nodos são pequenas
- Vamos analisar com cuidado!

Análise da complexidade de pior caso

- Inicialmente, relembramos que um *heap* de n elementos tem altura $\lfloor log_2(n) \rfloor$
- e o número de nodos de uma árvore qualquer de altura h é no máximo $\lceil \frac{n}{2^{h+1}} \rceil$

Esta última observação é a relação entre o número de nodos e a altura, dados a partir de $n = 2^{h+1} - 1$

Análise da complexidade de pior caso

- Logo, o limite superior de `build_max_heap` para o número de comparações entre elementos do vetor é (onde $\frac{n}{2}$ é o número de iterações do algoritmo):

$$C(n) = \sum_{h=0}^{log_2(n)} h.\lceil \frac{\frac{n}{2}}{2^{h+1}} \rceil$$

- Após alguma manipulação algébrica temos que:

$$C(n) = \sum_{h=0}^{log_2(n)} h.\lceil \frac{\frac{n}{2}}{2^{h+1}} \rceil \leq \frac{n}{2}. \sum_{h=0}^{log_2(n)} \frac{h}{2^h}$$

Análise da complexidade de pior caso

- Agora temos um limitante superior em termos desta somatória:

$$C(n) \leq \frac{n}{2} . \sum_{h=0}^{log_2(n)} \frac{h}{2^h}$$

Análise da complexidade de pior caso

- Para continuarmos, é preciso resgatar um resultado do Cálculo Diferencial e Integral

$$\sum_{h=0}^{\infty} k.x^k = \frac{x}{(1-x)^2}, |x| < 1$$

- Para o nosso caso, o ideal é tomar $x = \frac{1}{2} < 1$, o que resulta em:

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

Análise da complexidade de pior caso

- Bem, somar até $log_2(n)$ certamente é menor do que somar até o infinito
- Assim chegamos em:

$$C(n) \leq \frac{n}{2} \sum_{h=0}^{log_2(n)} \frac{h}{2^h} \leq \frac{n}{2} \sum_{h=0}^{\infty} \frac{h}{2^h} = 2\frac{n}{2} = n$$

Ou seja:

$$C(n) \leq n$$

Ou seja: *build_max_heap* tem custo de pior caso linear

Conclusão

- Este resultado é espetacular!
- Construir um *heap* de máximo a partir de um vetor qualquer tem custo linear
- Isto será particularmente importante na análise do algoritmo *heapsort* que veremos na próxima aula

Fim do tópico

- O conteúdo desta aula está no livro Cormen, Leiserson, Rivest e Stein, no capítulo 6, seção 6.3

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`