CI1056: Algoritmos e Estruturas de Dados II

Prof. Dr. Marcos Castilho

Departamento de Informática/UFPR

4 de dezembro de 2020

Resumo

Análise do algoritmo *max-heapify*

Objetivos da aula

- Apresentar a análise do algoritmo *max-heapify*, a relação de recorrência e sua solução

Relação de recorrência

- Os pontos de interesse são as duas comparações nas linhas 4 e 8 e a chamada recursiva na linha 12, pois são as linhas que fazem comparações entre elementos de v

```
1  max-heapify (v, i)
2      esq = esquerda (i)
3      dir = direita (i)
4      se esq <= n e v[esq] > v[i]
5          maior = esq
6      senao
7          maior = i
8      se dir <= n e v[dir] > v[maior]
9          maior = dir
10     se maior != i
11         troca v[i] com v[maior]
12         max-heapify (v, maior)
```

Observações

- O algoritmo inicia em um certo nodo *i* do *heap*
- No melhor caso, este nodo *i* já está em seu lugar certo e temos uma subárvore que já é um *heap* de máximo, mas o algoritmo tem que fazer duas comparações para confirmar.
- Logo, no melhor caso, temos 2 comparações
- E para o pior caso?

Observações

- A relação de recorrência deve depender do tamanho da entrada, isto é, do número de nodos da subárvore que está sendo processada
- Um caso especial é quando o algoritmo inicia na raiz do *heap*, isto é, quando $i = 1$
- Mas a entrada pode ser um nodo i que não necessariamente está na raiz
- Por isso vamos basear nossa análise na altura deste nodo com relação às folhas

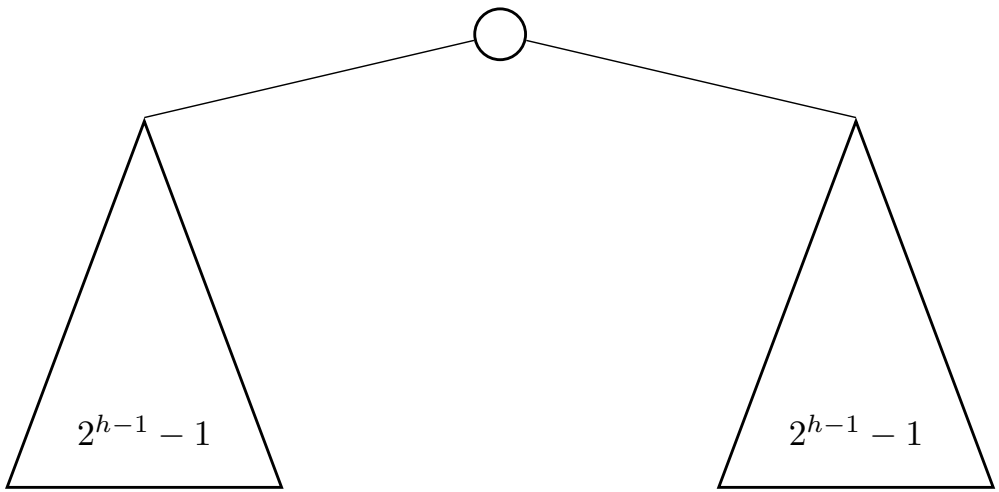
Altura de um nodo em função das folhas

Na nossa análise vamos considerar que um nodo i está na altura h com relação às folhas assim:

- Se i é um nodo folha, então $h = 0$
- Se i não é um nodo folha, então sua altura é o número de arestas entre este nodo i e a folha mais distante
- Para facilitar, vamos supor que temos uma árvore binária completa
- Se não for completa então a altura no lado esquerdo é uma unidade maior, o que não é grave em termos de complexidade, só atrapalha um pouco as contas

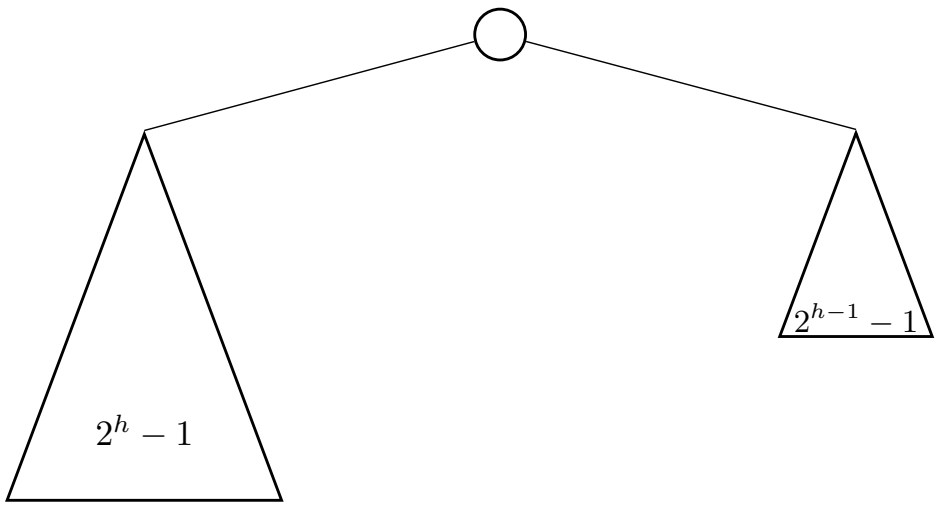
O caso da árvore completa

- Uma árvore binária completa de altura h tem exatamente $n = 2^h - 1$ nodos
- O número de nodos da subárvore da esquerda é igual ao da subárvore da direita, que é exatamente $2^{h-1} - 1$
- Evidentemente, a altura desta árvore pode ser calculada facilmente: $h = \lfloor log_2(n) \rfloor + 1$



O caso da árvore não completa

- Uma árvore binária não completa é desbalanceada no máximo em um nível, logo, se ela tem altura h, tem menos do que $n = 2^h - 1$ nodos
- Por exemplo, no caso abaixo, ela tem $n = 3.2^{h-1} - 1$ nodos
- Logo, $h = \lfloor log_2(n) \rfloor + 1$ é um limitante superior para a altura



Simplificação

- Vamos estudar o caso da árvore binária completa
- O algoritmo `max_heapify` processa uma subárvore iniciando em algum nodo i que pode estar em qualquer lugar da árvore
- O pior caso do algoritmo é quando todas as recursões são feitas até chegar em uma folha desta árvore binária completa, sendo que i é a raiz da árvore
- O melhor caso, como vimos, faz só duas comparações

A relação de recorrência em função da altura

$$C(h) \leq \begin{cases} 0, & \text{se } h = 0 \\ C(h-1) + 2, & \text{se } h > 0 \end{cases}$$

- Esta é uma relação de recorrência simples que pode ser resolvida pela conhecida técnica de "abrir" a relação
- A solução dela é: $C(h) = 2h$ (exercício)
- Então segue que $C(n) = 2.\lfloor log_2(n) \rfloor + 1$

Últimas observações

- Nossa análise foi feita supondo o pior caso, quando o algoritmo inicia na raiz
- Mas sabemos que o algoritmo pode iniciar em qualquer nodo da árvore, então o custo do algoritmo `max_heapify` depende essencialmente da altura deste nodo em relação às folhas
- O custo será sempre logaritmo em função desta altura
- Isto é importante quando analisarmos o algoritmo de construção de um *heap* que veremos em seguida

Fim do tópico

- O conteúdo desta aula está no livro Cormen, Leiserson, Rivest e Stein, no capítulo 6, seções 6.1 e 6.2. Também está no livro Sedgewick e Wayne, seção 2.4, conforme referência já citadas
- Na próxima aula veremos como se constrói um *heap* e como isso pode servir para um belo algoritmo de ordenação: o *heapsort*

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/`