Árvore Binária de Busca Ótima

Siang Wun Song - Universidade de São Paulo - IME/USP

MAC 5710 - Estruturas de Dados - 2008

Referência bibliográfica

Os slides sobre este assunto são parcialmente baseados nas seções sobre árvore binária de busca ótima do capítulo 4 do livro

- N. Wirth. Algorithms + Data Structures = Programs. Prentice Hall, 1976.

Árvore binária de busca ótima

- Há situações em que sabemos com antecedência quais as chaves que serão buscadas e, mais ainda, com que freqüência ou probabilidade cada chave será buscada.
- Nesse caso, podemos construir uma árvore binária de busca ótima que será eficiente para a operação de busca. Inserções e remoções não são usualmente efetuadas na árvore assim construída pois elas podem modificar a sua estrutura. Caso inserções e remoções são permitidas, então se deve periodicamente reconstruir a árvore ótima.
- A melhor árvore binária de busca depende das probabilidades de acesso das chaves. Quando a distribuição dessas probabilidades não é uniforme, a melhor árvore binária de busca não é necessáriamente uma árvore binária perfeitamente balanceada. Intuitivamente, queremos colocar as chaves mais buscadas mais próximas ao topo da árvore binária.
- Vamos estudar como constuir uma árvore binária de busca ótima. Mas, antes, temos que definir o que se entende por árvore binária de busca ótima.

Comprimento de caminho ponderado de uma árvore

- Denotamos por comprimento de caminho h_i de um nó k_i o número de nós encontrados desde a raiz até o nó k_i.
- Ele expressa o número de comparações realizadas para buscar uma chave no nó k_i.

Considere uma árvore binária de busca com n chaves

$$k_1 < k_2 < \ldots < k_n.$$

Suponha que se conhece a probabilidade de acesso de cada uma das chaves: sendo p_i a probabilidade de acesso à chave k_i, para $1 \leq i \leq n$:

$$\sum_{i=1}^{n} p_i = 1$$

Comprimento de caminho ponderado de uma árvore

O custo de busca P da árvore é expresso pelo comprimento de caminho ponderado da árvore, assim definida:

$$P = \sum_{i=1}^{n} p_i \, h_i$$

onde p_i é a probabilidade de acesso à chave k_i e h_i é o comprimento de caminho de k_i.

- Se toda chave tem igual probabilidade de ser buscada, então $p_i = 1/n$, $1 \leq i \leq n$ e teremos o comprimento de caminho médio da árvore, já visto anteriormente:

$$P = \sum_{i=1}^{n} p_i \, h_i = \sum_{i=1}^{n} \frac{h_i}{n} = \frac{1}{n} \sum_{i=1}^{n} h_i$$

Árvore binária de busca ótima

Considere árvores binárias de busca com n chaves

$$k_1 < k_2 < \ldots < k_n.$$

Seja p_i a probabilidade de acesso à chave k_i, para $1 \leq i \leq n$.

Dentre todas tais árvores, é dita árvore binária de busca ótima aquela que minimiza o custo P:

$$P = \sum_{i=1}^{n} p_i \, h_i$$

onde h_i é o comprimento de caminho de k_i.

Exemplo de uma árvore binária de busca ótima

Considere $n = 3$ e as 3 chaves

- $k_1 = 100$
- $k_2 = 200$
- $k_3 = 300$

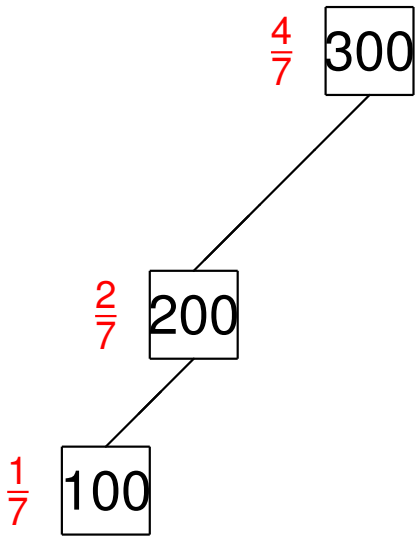
com as respectivas probabilidades de acesso

- $p_1 = \frac{1}{7}$
- $p_2 = \frac{2}{7}$
- $p_3 = \frac{4}{7}$.

Vamos ilustrar as possíveis árvores binárias de busca e seu respectivo custo P.

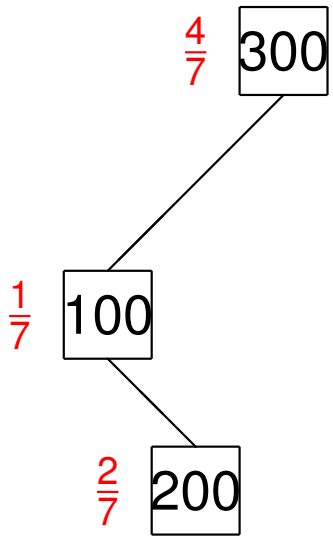
Árvore 1

Custo $P = 3 \times \frac{1}{7} + 2 \times \frac{2}{7} + 1 \times \frac{4}{7} = \frac{11}{7}$



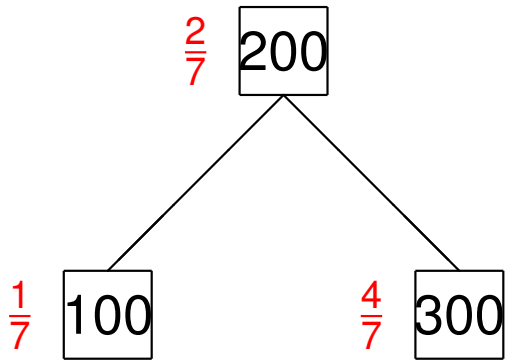
Árvore 2

Custo $P = 3 \times \frac{2}{7} + 2 \times \frac{1}{7} + 1 \times \frac{4}{7} = \frac{12}{7}$



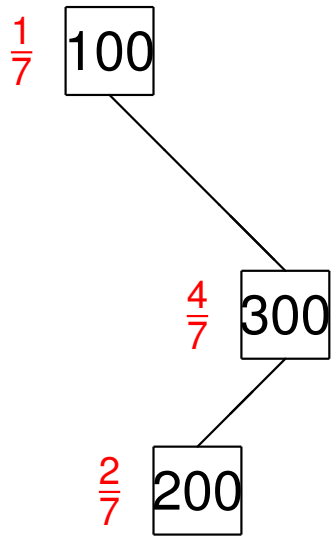
Árvore 3

Custo $P = 2 \times \frac{1}{7} + 2 \times \frac{4}{7} + 1 \times \frac{2}{7} = \frac{12}{7}$



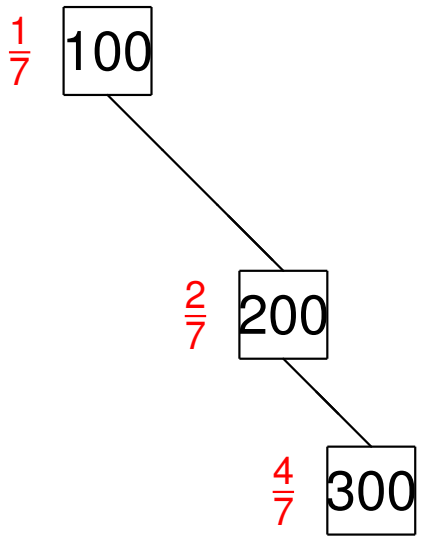
Árvore 4

Custo $P = 3 \times \frac{2}{7} + 2 \times \frac{4}{7} + 1 \times \frac{1}{7} = \frac{15}{7}$



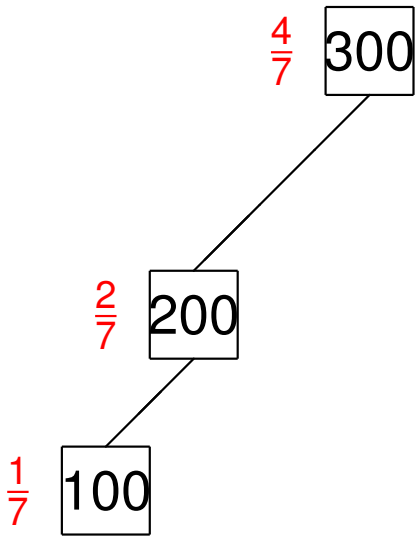
Árvore 5

Custo $P = 3 \times \frac{4}{7} + 2 \times \frac{2}{7} + 1 \times \frac{1}{7} = \frac{17}{7}$



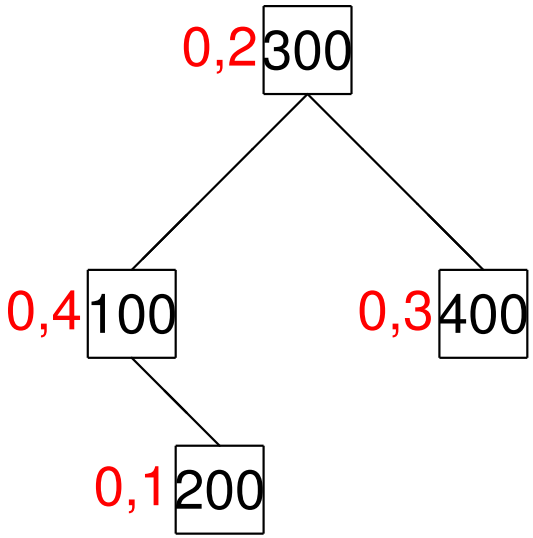
Obs. 1: A árvore ótima não sempre é balanceada

Árvore 1 é a árvore ótima e não é balanceada.



Obs.2: A estratégia gulosa não funciona

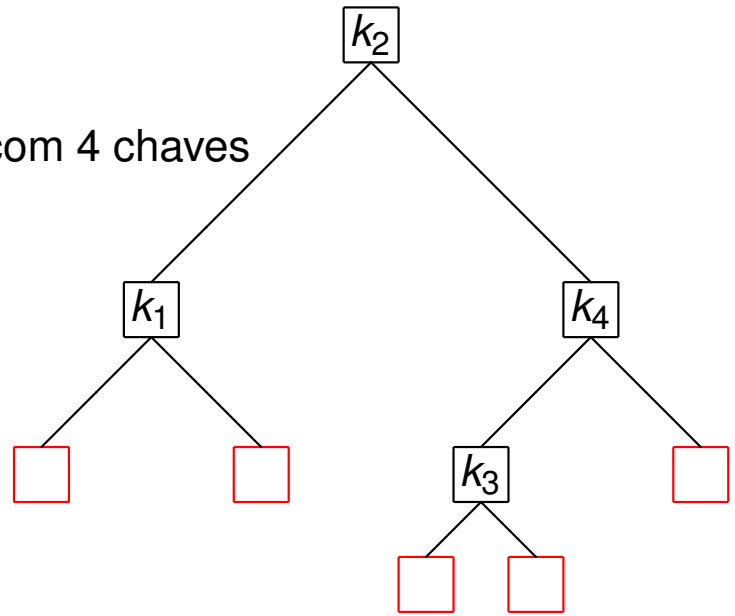
- Pode parecer que basta adotar uma estratégia gulosa e começar colocando a chave de maior probabilidade de acesso na raiz.
- O exemplo mostra uma árvore ótima com as probabilidades de acesso em vermelho.
- A chave de maior probabababilidade de acesso não está na raiz.



Busca de chaves ausentes na árvore

- Pode-se estranhar por que sempre a chave buscada tem que estar na árvore. Muitas vezes, o que procuramos não está na árvore.
- Quando buscamos uma chave ausente, caímos num ponteiro nil. Na figura, representamos pela cor vermelha os nós fictícios em que caem as buscas de chaves ausentes.
- Nessa formulação mais geral, nós internos representam chaves presentes e as folhas (nós fictícios vermelhos) representam intervalos de chaves ausentes.

Uma árvore com 4 chaves k_1, k_2, k_3 e k_4



Uma formulação mais geral de árvore ótima

Considere árvores binárias de busca com n chaves $k_1 < k_2 < \ldots < k_n$.

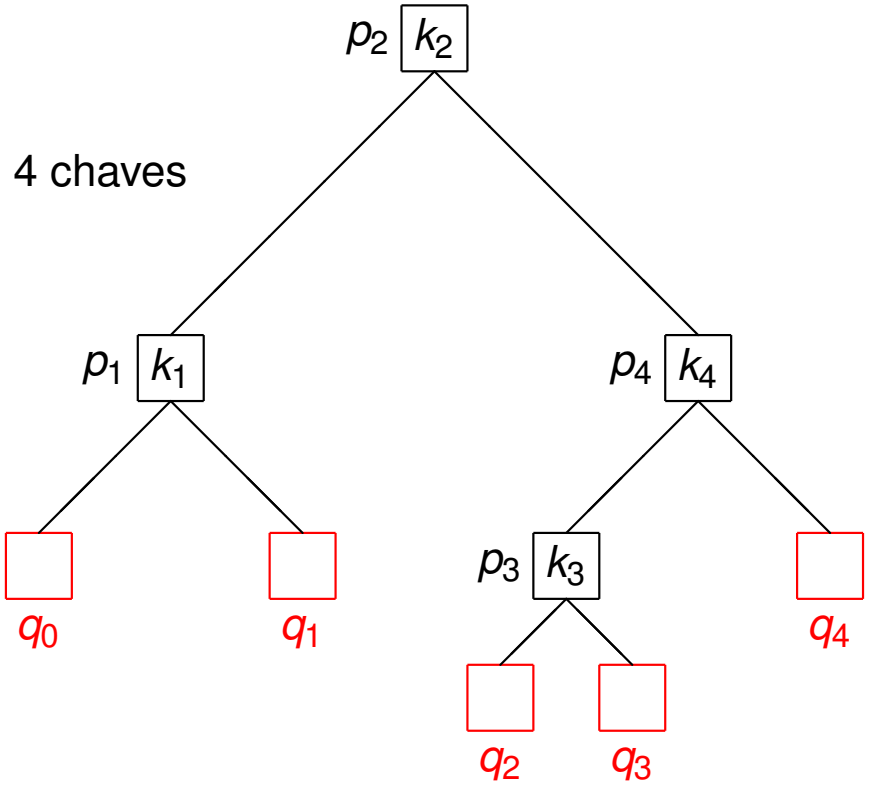
- Seja h_i o comprimento de caminho da chave k_i, $1 \leq i \leq n$.
- Conhecemos as probabilidades, p_i e q_i, com $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$, onde
  - p_i = a probabilidade de acesso à chave k_i, para $1 \leq i \leq n$.
  - q_0 = a probabilidade de acesso a toda chave x, $x < k_1$.
  - q_n = a probabilidade de acesso a toda chave x, $x > k_n$.
  - q_i = a probabilidade de acesso a toda chave x, $k_i < x < k_{i+1}$, $1 \leq i < n$.

Uma árvore binária de busca ótima é aquela que minimiza o custo P, onde h'_i representa o comprimento de caminho do nó fictício em que caem buscas sem sucesso.

$$P = \sum_{i=1}^{n} p_i \, h_i + \sum_{i=0}^{n} q_i \, h'_i$$

Para o exemplo da árvore com 4 chaves

Uma árvore com 4 chaves k_1, k_2, k_3 e k_4



Uma árvore binária de busca ótima minimiza o custo P:

$$P = \sum_{i=1}^{4} p_i \; h_i + \sum_{i=0}^{4} q_i \; h'_i$$

Uso de freqüências no lugar de probabilidades

Freqüências de acesso podem ser levantadas experimentalmente. Um modo equivalente é usar as freqüências no lugar das probabilidades.

Sejam freqüências de acesso, a_i e b_i, com $\sum_{i=1}^{n} a_i + \sum_{i=0}^{n} b_i = W$, onde

- a_i = a freqüência de acesso à chave k_i, $1 \leq i \leq n$.
- b_0 = a freqüência de acesso a toda chave x, $x < k_1$.
- b_n = a freqüência de acesso a toda chave x, $x > k_n$.
- b_i = a freqüência de acesso a toda chave x, $k_i < x < k_{i+1}$, $1 \leq i < n$.

Temos: $p_i = a_i/W$, $1 \leq i \leq n$ e $q_i = b_i/W$, $0 \leq i \leq n$.
A árvore binária de busca ótima então é aquela que minimiza

$$P = \sum_{i=1}^{n} a_i \; h_i + \sum_{i=0}^{n} b_i \; h'_i$$

Como obter uma árvore binária de busca ótima

No exemplo para $n = 3$ chaves, enumeramos todas as 5 árvores binárias possíveis e depois obtivemos a ótima.
Pergunta: este método é viável? Qual a sua complexidade de tempo?

Dado n, o número de árvores binárias possíveis com n nós é um número chamado número Catalan que vale:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

n	C_n	n	C_n
0	1	13	742900
1	1	14	2674440
2	2	15	9694845
3	5	16	35357670
4	14	17	129644790
5	42	18	477638700
6	132	19	1767263190
7	429	20	6564120420
8	1430	21	24466267020
9	4862	22	91482563640
10	16796	23	343059613650
11	58786	24	1289904147324
12	208012	25	4861946401452

Como obter uma árvore binária de busca ótima

No exemplo para $n = 3$ chaves, enumeramos todas as 5 árvores binárias possíveis e depois obtivemos a ótima.
Pergunta: este método é viável? Qual a sua complexidade de tempo?

Dado n, o número de árvores binárias possíveis com n nós é um número chamado número Catalan que vale:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

n	C_n	n	C_n
0	1	13	742900
1	1	14	2674440
2	2	15	9694845
3	5	16	35357670
4	14	17	129644790
5	42	18	477638700
6	132	19	1767263190
7	429	20	6564120420
8	1430	21	24466267020
9	4862	22	91482563640
10	16796	23	343059613650
11	58786	24	1289904147324
12	208012	25	4861946401452

Como obter uma árvore binária de busca ótima

No exemplo para $n = 3$ chaves, enumeramos todas as 5 árvores binárias possíveis e depois obtivemos a ótima.
Pergunta: este método é viável? Qual a sua complexidade de tempo?

Dado n, o número de árvores binárias possíveis com n nós é um número chamado número Catalan que vale:

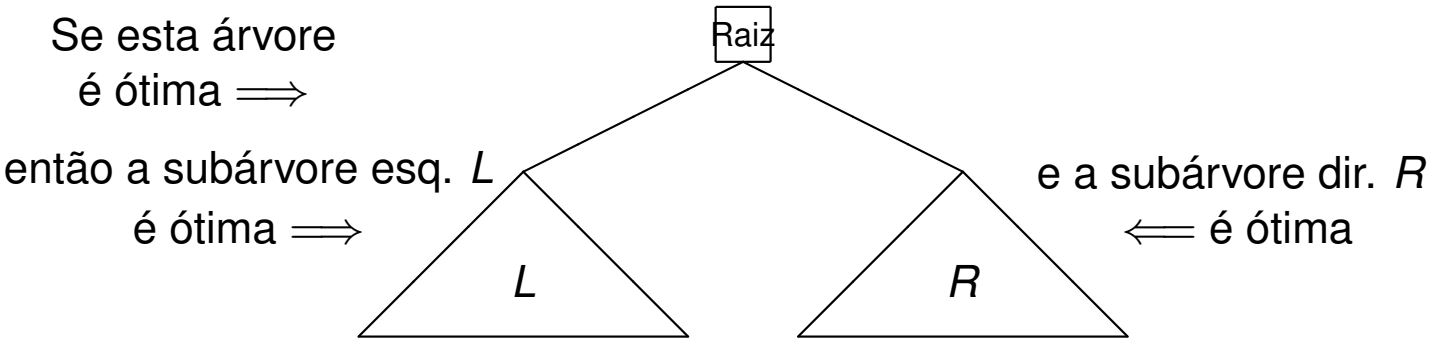
$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

n	C_n	n	C_n
0	1	13	742900
1	1	14	2674440
2	2	15	9694845
3	5	16	35357670
4	14	17	129644790
5	42	18	477638700
6	132	19	1767263190
7	429	20	6564120420
8	1430	21	24466267020
9	4862	22	91482563640
10	16796	23	343059613650
11	58786	24	1289904147324
12	208012	25	4861946401452

Método de programação dinâmica

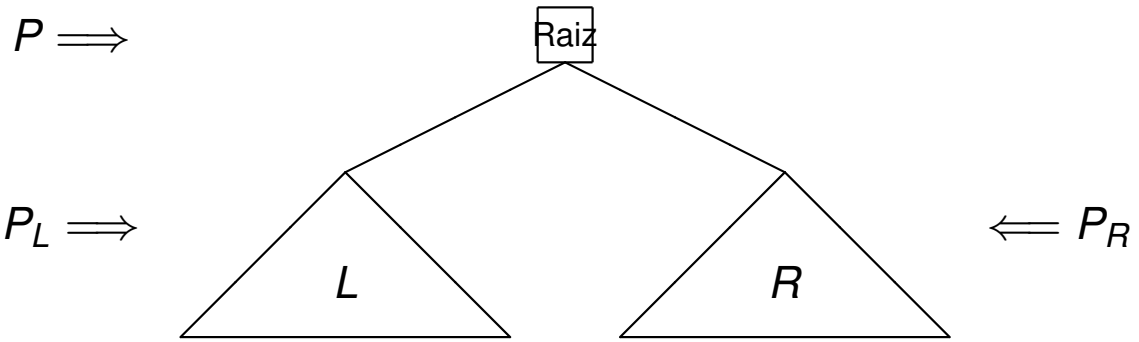
- Em alguns algoritmos, a ineficiência se deve ao não reuso de resultados já calculados anteriormente.
- A idéia básica do método de programação dinâmica é o armazenamento e uso de soluções ótimas de subproblemas para obter a solução ótima do problema geral.

Propriedade de árvores binárias de busca ótimas:
Se uma árvore binária de busca é ótima, então as suas subárvores são ótimas.



Expressão de P em termos de P_L e P_R

Seja $W = \sum_{i=1}^{n} a_i + \sum_{i=0}^{n} b_i$ denominado o peso da árvore, i.e. o número total de buscas efetuadas, incluindo as com ou sem sucesso.

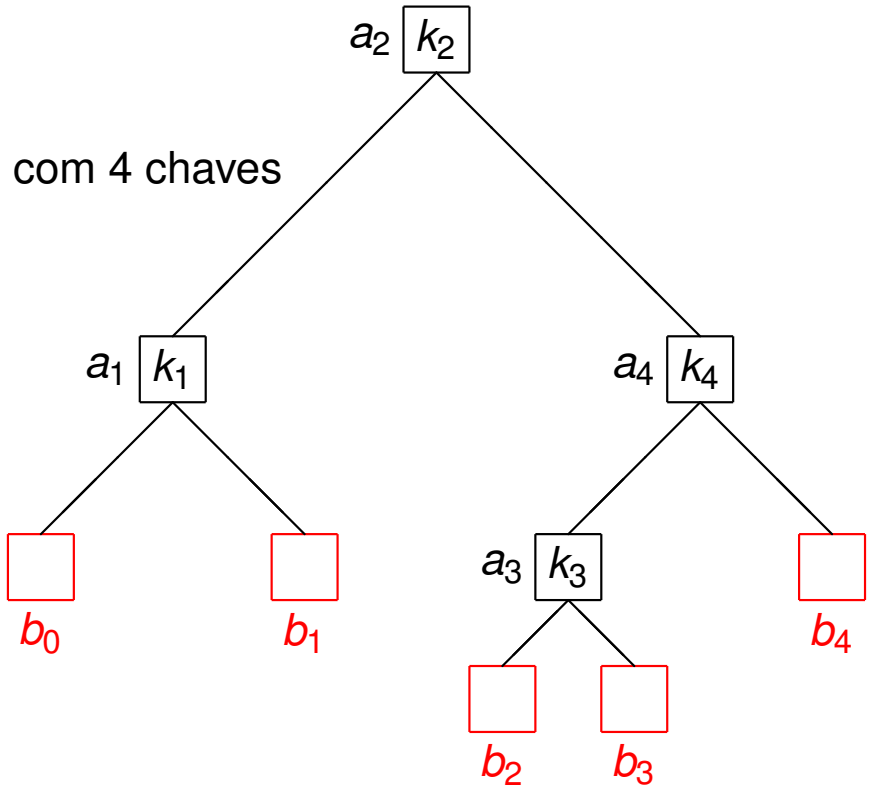


Qualquer busca que segue para L ou para R passa pela raiz.

Temos portanto $P = P_L + W + P_R$.

Exemplo que ilustra a relação $P = P_L + W + P_R$

Seja uma árvore com 4 chaves k_1, k_2, k_3 e k_4



$$P = 2a_1 + a_2 + 3a_3 + 2a_4 + 3b_0 + 3b_1 + 4b_2 + 4b_3 + 3b_4$$

$$P_L = a_1 + 2b_0 + 2b_1 \text{ e } P_R = 2a_3 + a_4 + 3b_2 + 3b_3 + 2b_4$$

$$P = P_L + \underbrace{(a_1 + a_2 + a_3 + a_4)}_{\sum_{i=1}^{n} a_i} + \underbrace{(b_0 + b_1 + b_2 + b_3 + b_4)}_{\sum_{i=0}^{n} b_i} + P_R$$

$W = \sum_{i=1}^{n} a_i + \sum_{i=0}^{n} b_i$ e o exemplo ilustra $P = P_L + W + P_R$.

Construção de uma árvore binária de busca ótima

- O método de programação dinâmica constrói uma árvore ótima a partir de subárvores ótimas. Vamos ilustrar o método com um pequeno exemplo.
- Sem perda de generalidade, vamos usar a formulação mais simples, sem considerar frequências de buscas de chaves ausentes.

Sejam $n = 5$ chaves valendo 1, 2, 3, 4 e 5 com as respectivas freqüências de acesso:

chave	1	2	3	4	5
freq.	10	4	20	16	2

Vamos construir uma árvore binária de busca ótima de 5 nós com as 5 chaves dadas.

Árvores com uma chave

Chave k_i	1	2	3	4	5
Freq. a_i	10	4	20	16	2

- Chave 1:

A árvore ótima só tem a raiz onde fica a chave 1. $P = 10$.

- Chave 2:

A árvore ótima só tem a raiz onde fica a chave 2. $P = 4$.

- Chave 3:

A árvore ótima só tem a raiz onde fica a chave 3. $P = 20$.

- Chave 4:

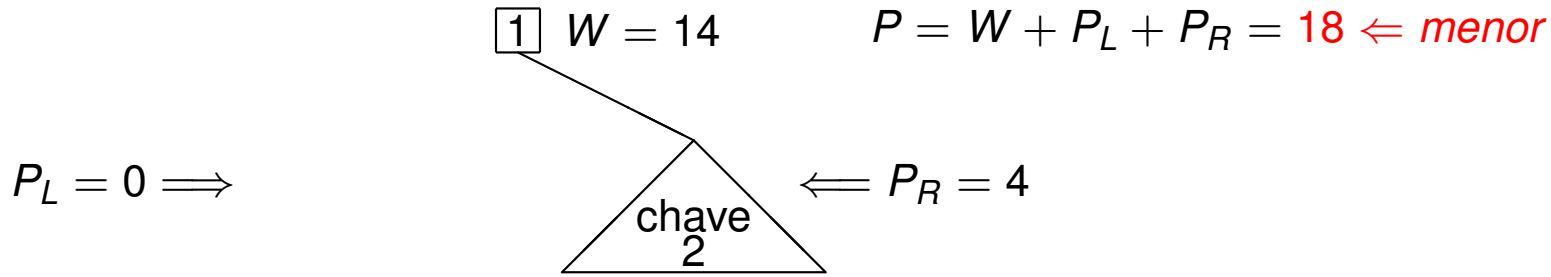
A árvore ótima só tem a raiz onde fica a chave 4. $P = 16$.

- Chave 5:

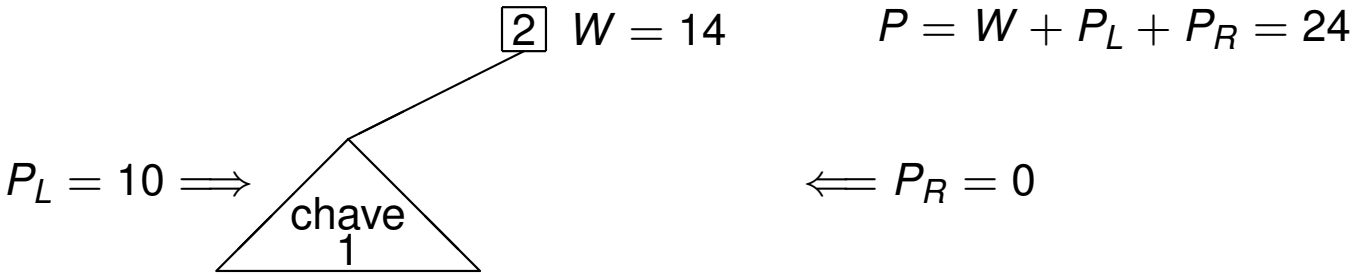
A árvore ótima só tem a raiz onde fica a chave 5. $P = 2$.

Árvores com duas chaves

- Chaves 1 e 2:
  - 1 na raiz.

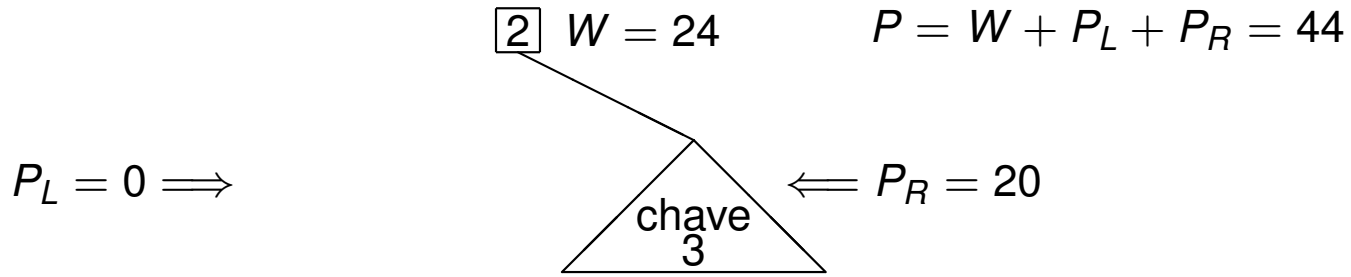


  - 2 na raiz.

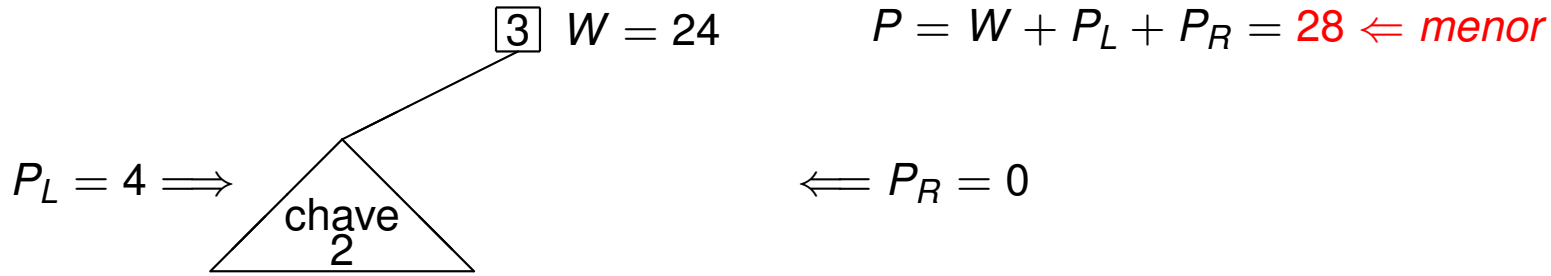


Árvores com duas chaves

- Chaves 2 e 3:
  - 2 na raiz.

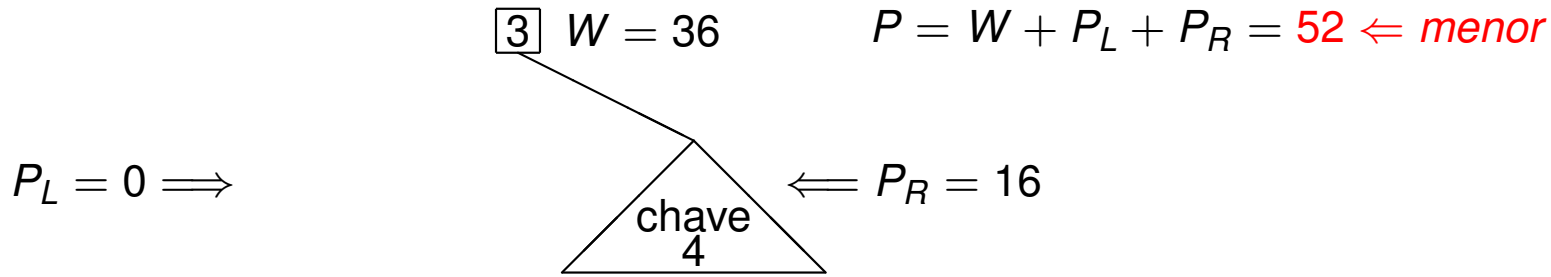


  - 3 na raiz.

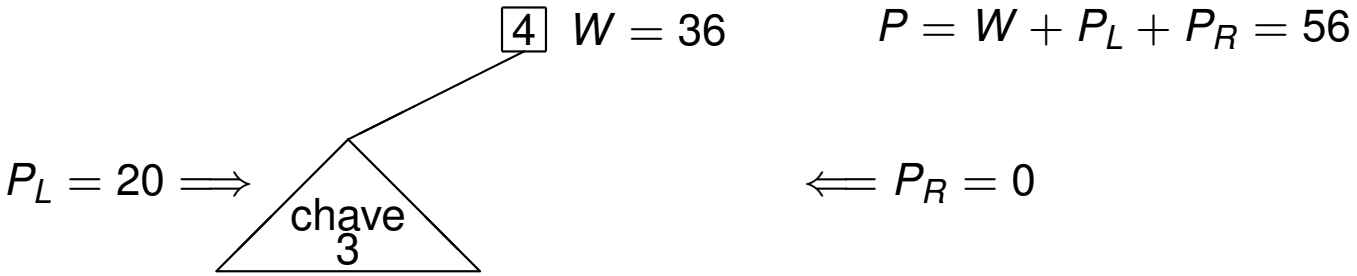


Árvores com duas chaves

- Chaves 3 e 4:
  - 3 na raiz.

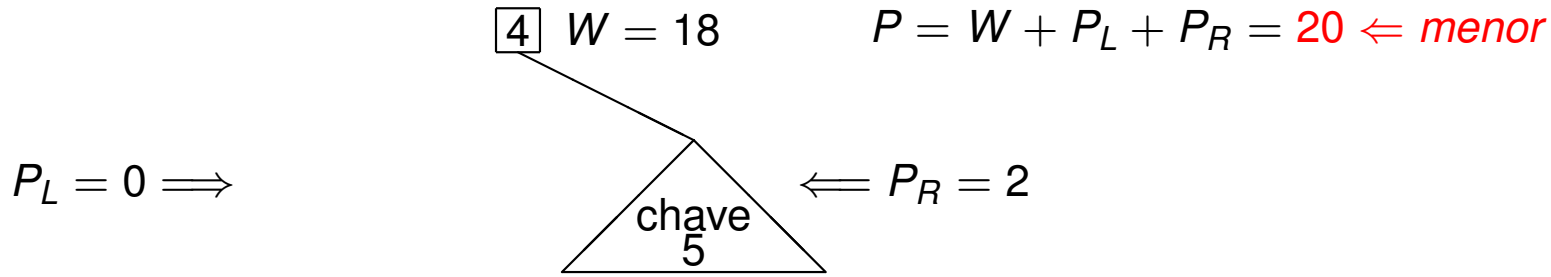


  - 4 na raiz.

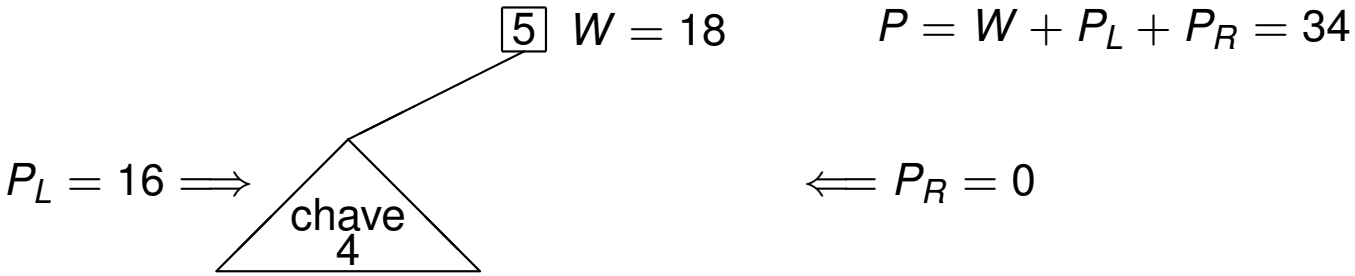


Árvores com duas chaves

- Chaves 4 e 5:
  - 4 na raiz.

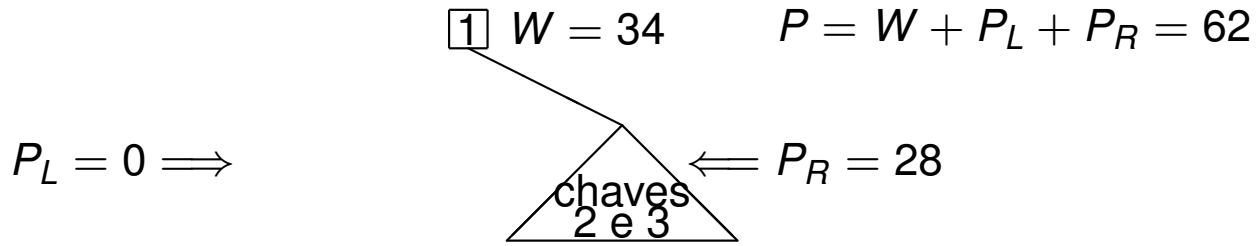


  - 5 na raiz.

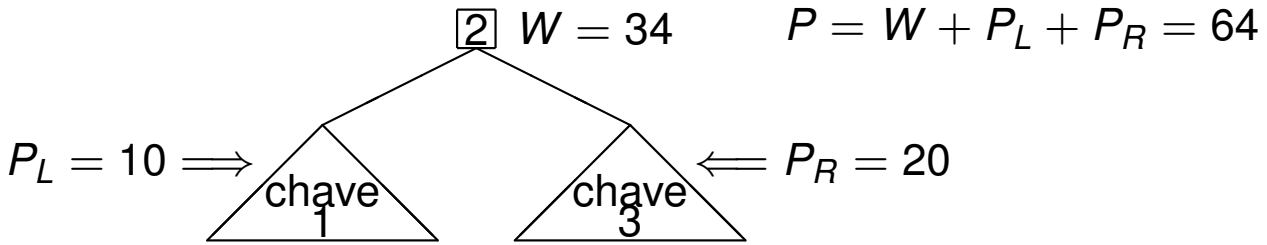


Árvores com três chaves

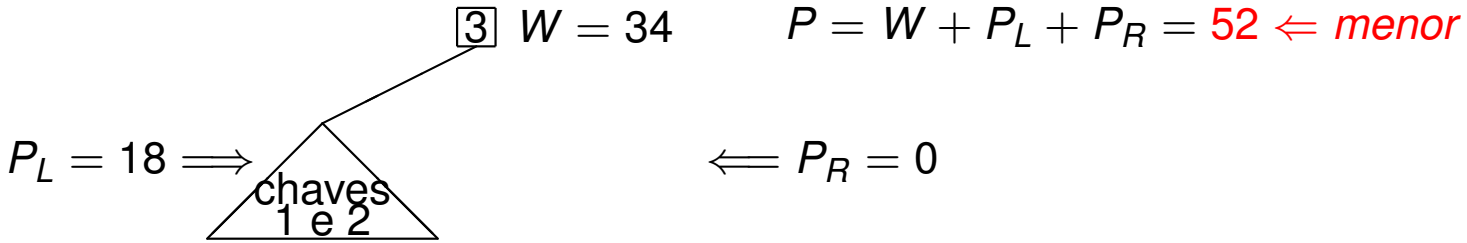
- Chaves 1, 2 e 3:
  - 1 na raiz.



  - 2 na raiz.

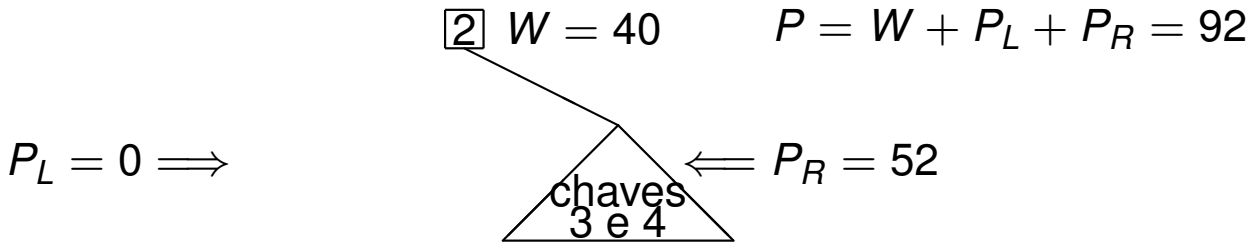


  - 3 na raiz.

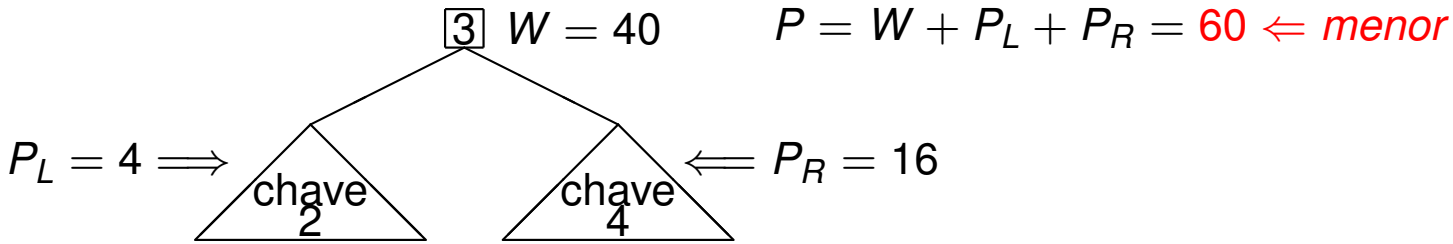


Árvores com três chaves

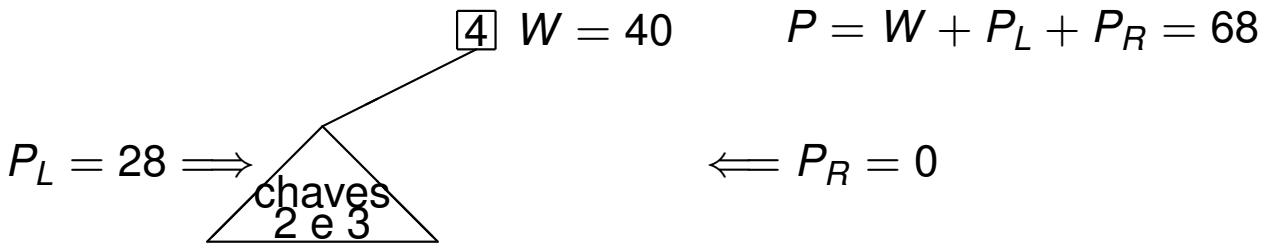
- Chaves 2, 3 e 4:
  - 2 na raiz.



  - 3 na raiz.

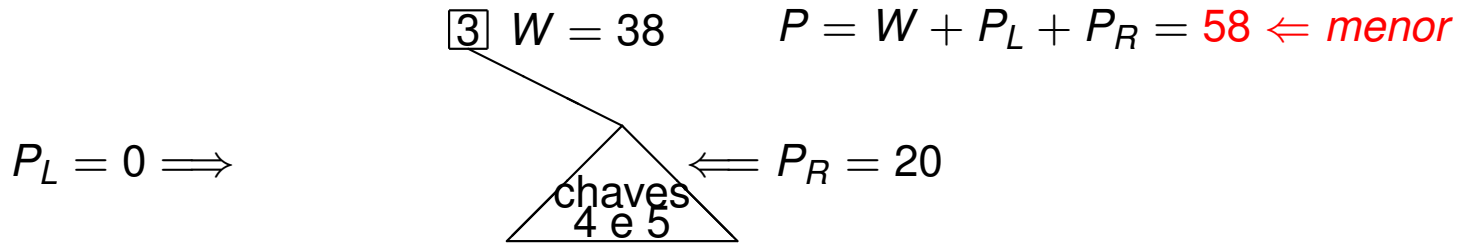


  - 4 na raiz.

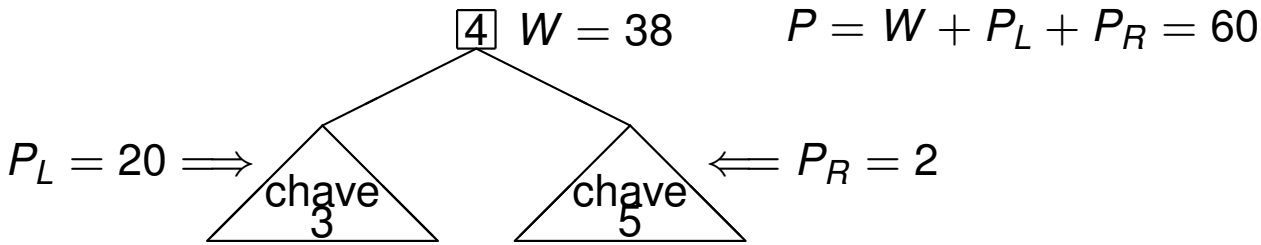


Árvores com três chaves

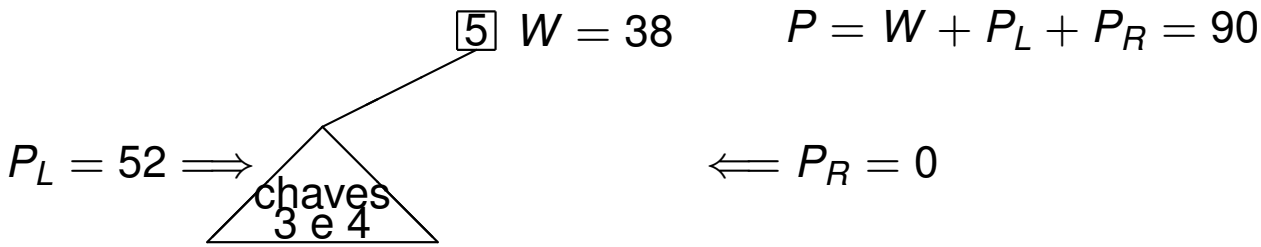
- Chaves 3, 4 e 5:
  - 3 na raiz.



  - 4 na raiz.

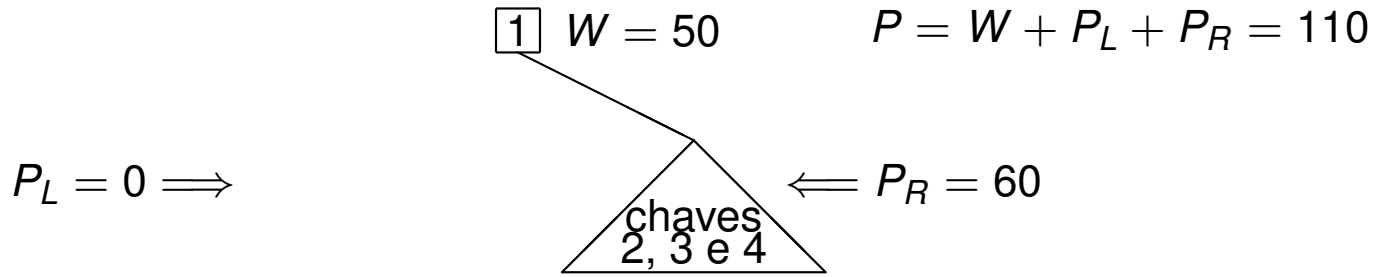


  - 5 na raiz.

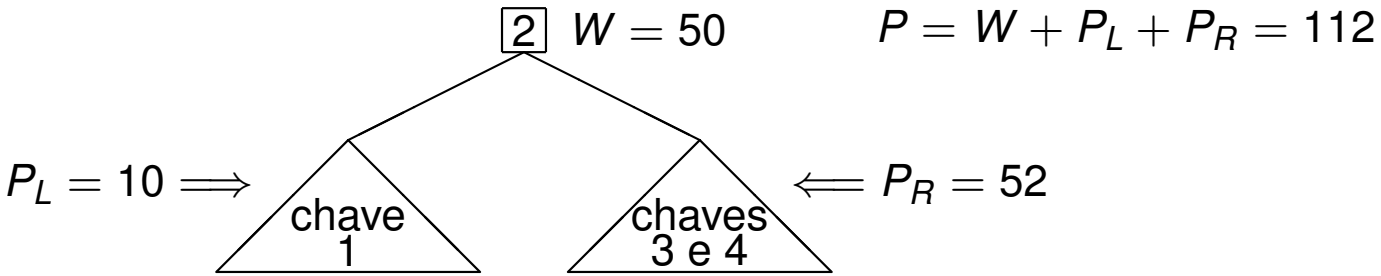


Árvores com quatro chaves

- Chaves 1, 2, 3 e 4:
  - 1 na raiz.

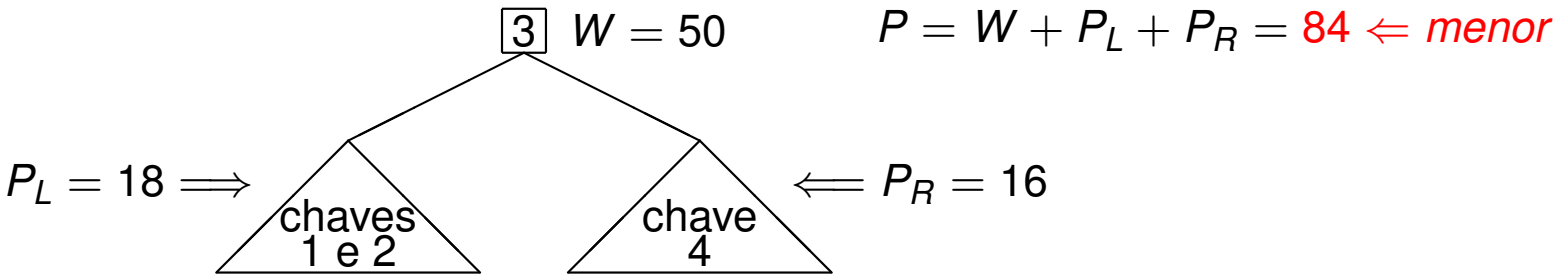


  - 2 na raiz.

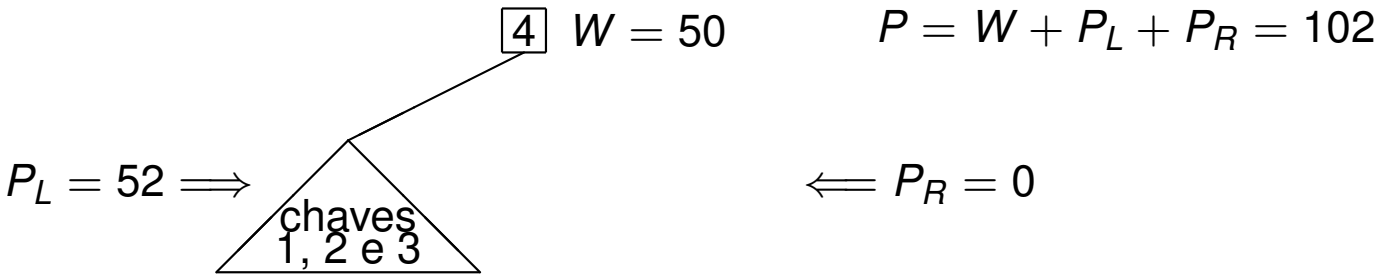


Árvores com quatro chaves

- Chaves 1, 2, 3 e 4:
  - 3 na raiz.

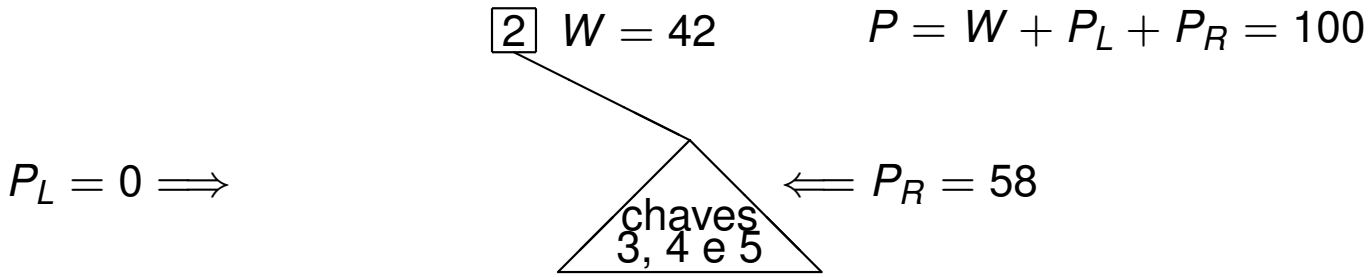


  - 4 na raiz.

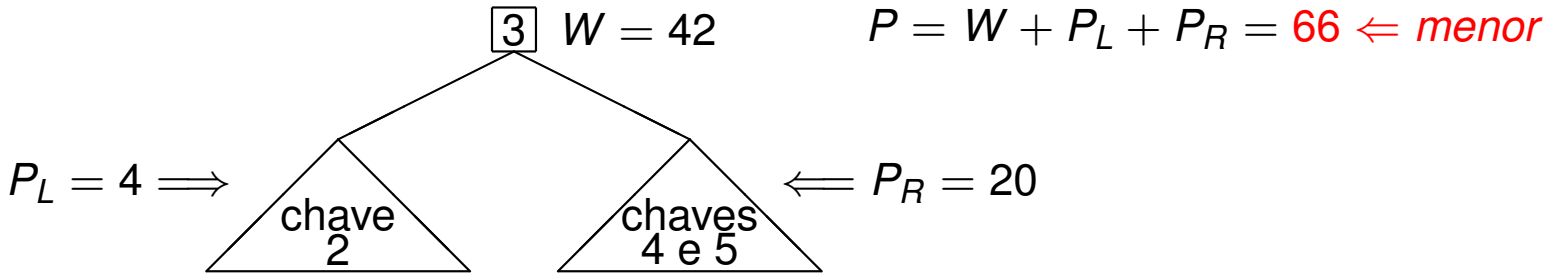


Árvores com quatro chaves

- Chaves 2, 3, 4 e 5:
  - 2 na raiz.

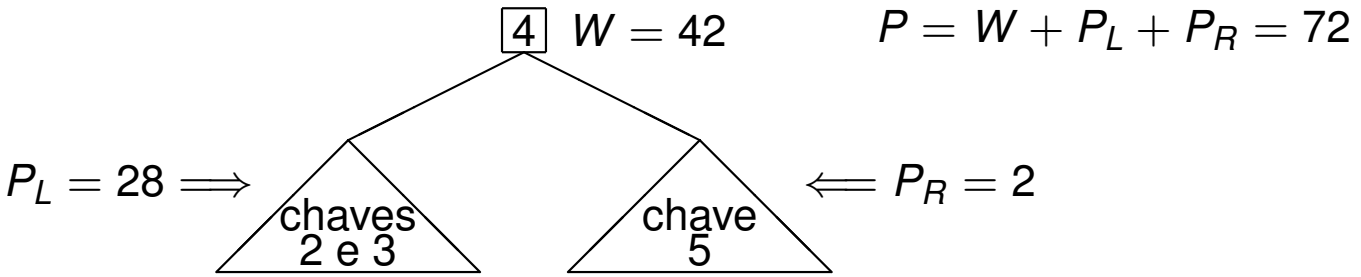


  - 3 na raiz.

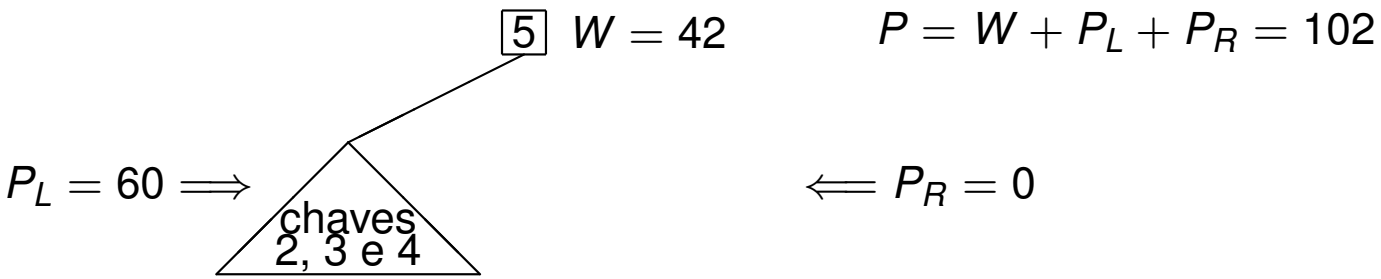


Árvores com quatro chaves

- Chaves 2, 3, 4 e 5:
  - 4 na raiz.

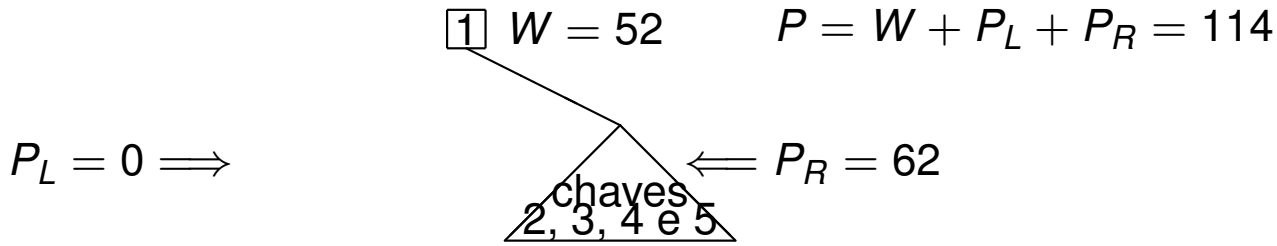


  - 5 na raiz.

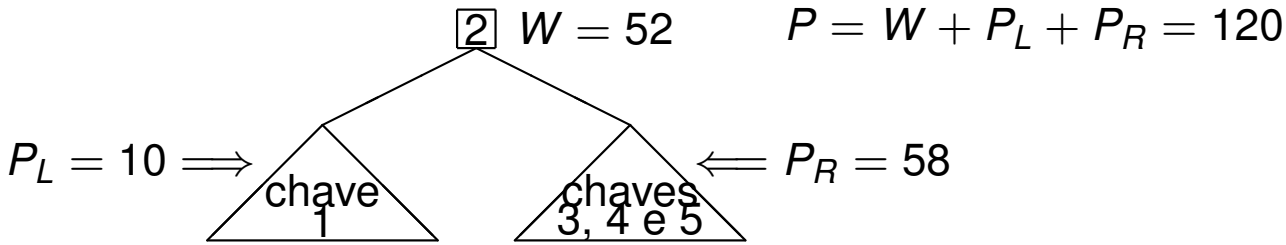


Árvores com cinco chaves

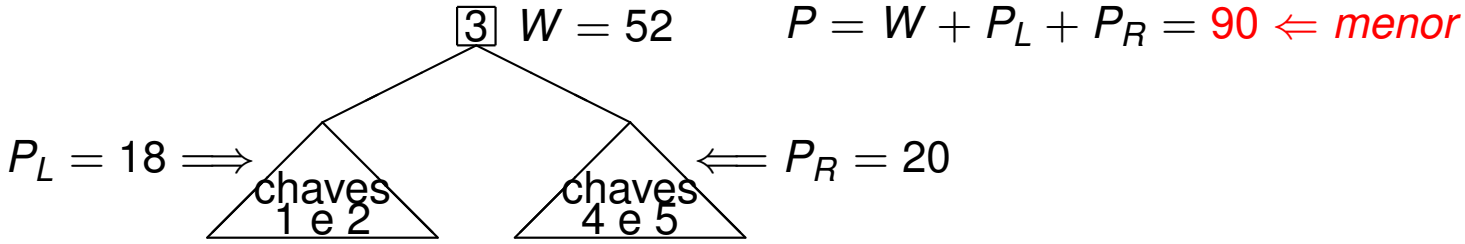
- Chaves 1, 2, 3, 4 e 5:
  - 1 na raiz.



  - 2 na raiz.

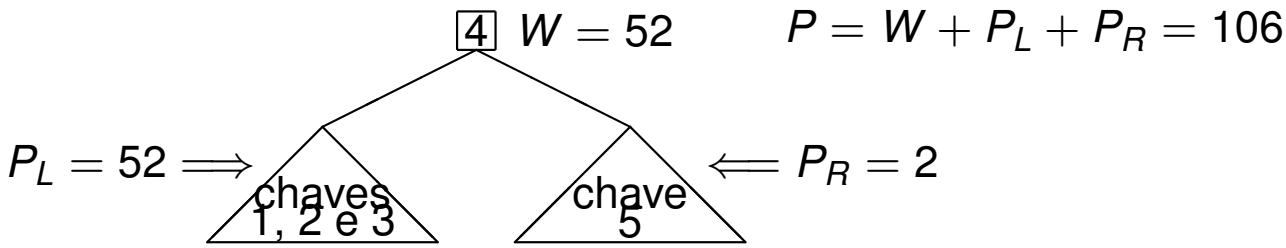


  - 3 na raiz.

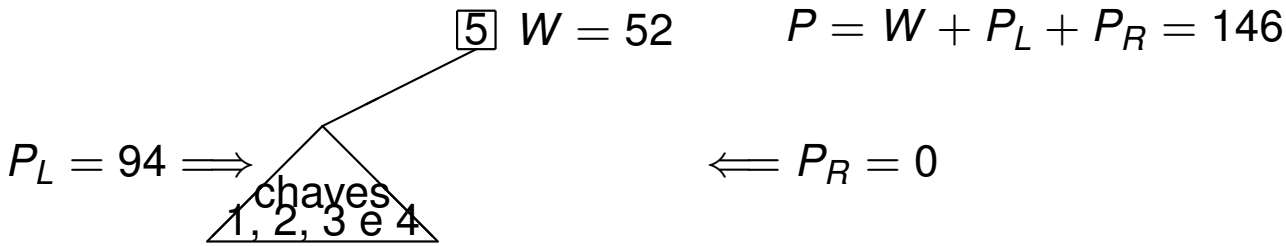


Árvores com cinco chaves

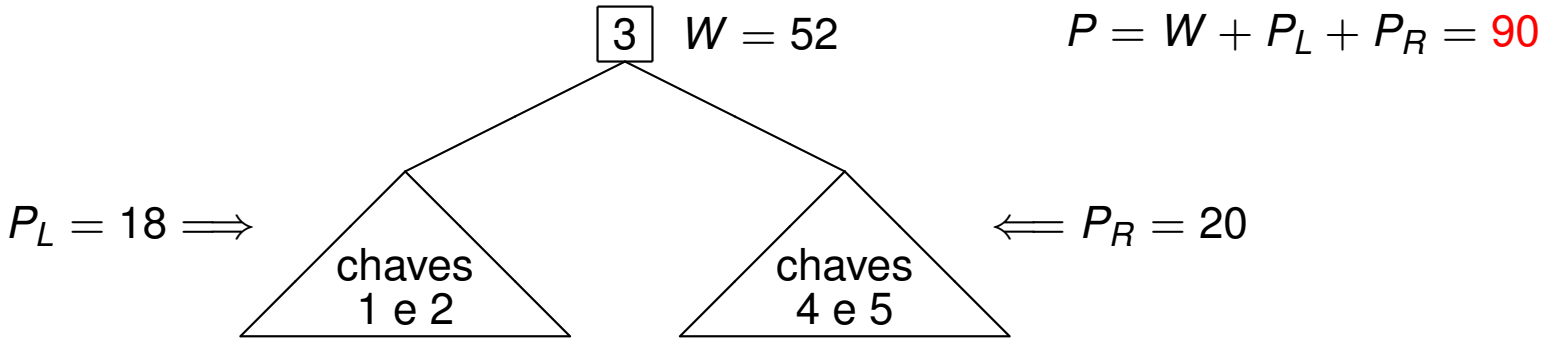
- Chaves 1, 2, 3, 4 e 5:
  - 4 na raiz.



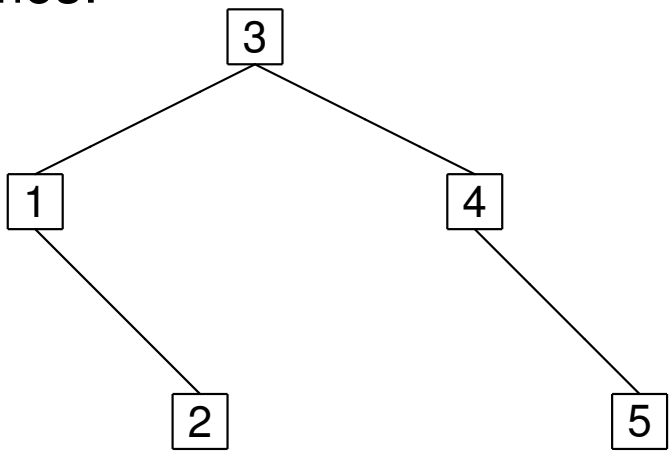
  - 5 na raiz.



A árvore ótima obtida



Basta recuperar a árvore ótima com chaves 1 e 2 e a árvore ótima com chaves 4 e 5. Temos:



Pode-se ver que o papel de W na quantidade P a ser minimizada é nulo, já que W entra em todas as comparações. Então podemos dispensar o uso de W na minimização de P.

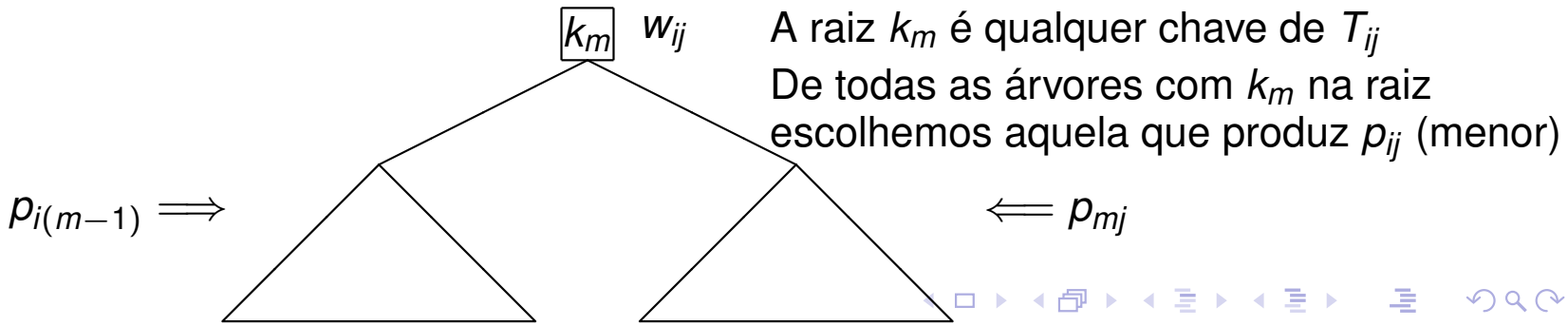
Algoritmo para construir uma árvore de busca ótima

Sejam dadas n chaves $k_1 < k_2 < \ldots < k_n$, com suas freq. de busca a_i de chaves presentes e freq. de busca de b_i de chaves ausentes, conforme já definidas.

Denote por T_{ij} uma árvore binária de busca ótima com as chaves $k_{i+1}, k_{i+2}, \ldots, k_j$, com peso w_{ij} e caminho p_{ij} definidos assim:

- $w_{ii} = b_i$, $0 \leq i \leq n$
- $w_{ij} = w_{i(j-1)} + a_j + b_j$, $0 \leq i < j \leq n$
- $p_{ii} = w_{ii}$, $0 \leq i \leq n$
- $p_{ij} = w_{ij} + \min_{i<m\leq j}(p_{i(m-1)} + p_{mj})$, $0 \leq i < j \leq n$.

A finalidade é obter T_{0n}, que contém as chaves $k_1, k_2, \ldots, k_n$.



Algoritmo para construir uma árvore de busca ótima

Seja T_{ij} uma árvore binária de busca ótima com as chaves $k_{i+1}, k_{i+2}, \ldots, k_j$.

A finalidade é obter T_{0n}.

Seja $h = j - i$, $i \leq j$. Obtemos T_{0n} a partir de árvores ótimas menores. Exemplo para $n = 5$:

$$\begin{matrix} T_{05} \\ h = 5 \end{matrix} \Leftarrow \begin{matrix} T_{04} \\ T_{15} \\ h = 4 \end{matrix} \Leftarrow \begin{matrix} T_{03} \\ T_{14} \\ T_{25} \\ h = 3 \end{matrix} \Leftarrow \begin{matrix} T_{02} \\ T_{13} \\ T_{24} \\ T_{35} \\ h = 2 \end{matrix} \Leftarrow \begin{matrix} T_{01} \\ T_{12} \\ T_{23} \\ T_{34} \\ T_{45} \\ h = 1 \end{matrix} \Leftarrow \begin{matrix} T_{00} \\ T_{11} \\ T_{22} \\ T_{33} \\ T_{44} \\ T_{55} \\ h = 0 \end{matrix}$$

Algoritmo para construir uma árvore de busca ótima

Dadas as freq. a_i e b_i, calculam-se os pesos w_{ij}:
$w_{ii} = b_i$, $0 \leq i \leq n$
$w_{ij} = w_{i(j-1)} + a_j + b_j$, $0 \leq i < j \leq n$.

Calculam-se p_{ij}:
$p_{ii} = w_{ii}$, $0 \leq i \leq n$
$p_{ij} = w_{ij} + \min_{i<m\leq j}(p_{i(m-1)} + p_{mj})$, $0 \leq i < j \leq n$.

Vamos ainda usar r_{ij} para guardar a posição m que produz o mínimo p_{ij}.

Algoritmo para construir uma árvore de busca ótima

1: **for** $i \leftarrow 0$ to n **do**
2: $\quad p_{ii} \leftarrow b_i$
3: **end for**
4: **for** $i \leftarrow 0$ to $n-1$ **do**
5: $\quad j \leftarrow i+1$
6: $\quad p_{ij} \leftarrow w_{ij} + pij + p_{jj}$
7: $\quad r_{ij} \leftarrow j$
8: **end for**
9: **for** $h \leftarrow 2$ to n **do**
10: $\quad$ **for** $i \leftarrow 0$ to $n-h$ **do**
11: $\quad\quad j \leftarrow i+h$
12: $\quad\quad$ Achar$m$ e $min = \min_{1<m\leq j}(p_{i(m-1)} + p_{mj})$
13: $\quad\quad p_{ij} \leftarrow min + w_{ij}$
14: $\quad\quad r_{ij} \leftarrow m$
15: $\quad$ **end for**
16: **end for**

A complexidade de tempo do algoritmo de construção é $O(n^3)$, mas pode ser reduzido para $O(n^2)$ (Knuth - The Art of Computer Programming Vol. 3).

Uso da técnica de programação dinâmica

O termo programação dinâmica foi usado por Richard Bellman. A utlizada dessa técnica pode ser mostrada em várias outras aplicações.

- Encontrar uma ordem de multiplicar matrizes $M_1 M_2 \ldots M_n$ que usa o menor número de operações escalares.
- Encontrar o menor caminho entre dois vértices de um grafo.
- Dadas duas seqüências de caracteres, transformar uma seqüência na outra usando operações de edição como substituição, inserção ou remoção.

Essas aplicações são tratadas em cursos de Análise de Algoritmos.