Árvores de Huffman

Caio César Silva Souza e Airton Bordin Junior

Instituto de Informática
Universidade Federal de Goiás

Pós-Graduação em Ciência da Computação, 2017

Conceitos

- Método de compressão que faz uso das probabilidades de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de representação para cada símbolo
- Códigos de tamanho variável
  - Símbolos mais comuns possuem códigos menores na Árvore de Huffman
  - Por consequência, símbolos menos comum possuem códificações maiores

Conceitos

- Desenvolvido por David A. Huffman, estudante de doutorado no MIT, em 1952
- Publicado no artigo *A Method for the Construction of Minimum-Redundancy Codes*

Conceitos

- O algoritmo de Huffman tem como objetivo principal a compressão de dados
- Características importantes
  - Compressões podem variar de 20 a 90 porcento
  - É um algoritmo guloso que cria soluções dependendo das entradas de dados a ser manipuladas
  - Utiliza código de comprimento variável para representar dados frequentemente acessados

Conceitos

- Suponhamos um arquivo de dados de 100.000 caracteres (dentre 6 diferentes) que desejamos armazenar de forma compacta
- Observamos que os caracteres no arquivo ocorrem com as seguintes freqüências

Caractere	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5

Tabela: 1

Frequências dos caracteres no texto

Comprimento de Palavras

- Utilizando um código de caracteres binários
  - Podemos usar códigos de comprimento fixo ou variável
    - Conforme a representação abaixo, o código de comprimento fixo utiliza 300.000 bits, seguindo a seguinte lógica (3 bits por caractere):
    - (45x3 + 13x3 + 12x3 + 16x3 + 9x3 + 5x3) x 1.000 = 300.000

Caractere	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Palavras em comprimento Fixo	000	001	010	011	100	101

Tabela: 2

Frequências e representação fixa dos caracteres

Árvore de Huffman

Etapas do algoritmo de Huffman

- Levando em consideração o exemplo da tabela 1
- Conjunto inicial de n=6 nós, um para cada letra
- Estágios de montagem da árvore
  - Os menores elementos são separados e seu nó pai e criado como a soma dos filhos
  - Fila de prioridade é iniciada do menor elemento ao maior
  - Utiliza o conceito de Heap de mínimo

Árvore de Huffman

Heaps

- Árvores Binárias Completas
- Para todos os nós, exceto a raiz, o valor do pai é menor ou igual o valor do nó (heap de mínimo) ou o contrário (heap de máximo)

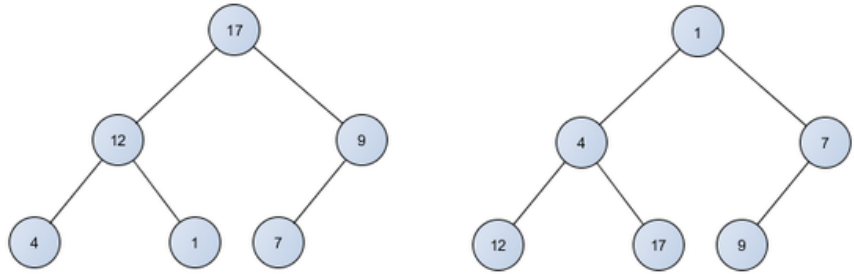


Figura: 1

Heap de máximo (esquerda) e Heap de mínimo (direita)

Árvore de Huffman

Heaps (definição formal)

- Uma árvore binária não-vazia é um heap de mínimo
  - Chave de busca associada à raiz é menor ou igual às chaves presentes em qualquer uma de suas sub-árvores
  - Ambas sub-árvores (se existirem) também são heaps
- Importante
  - Único nó é um heap
  - Todas as chaves de qualquer sub-árvore são maiores do que a chave presente no nó raiz

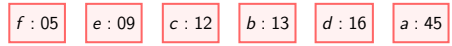
Árvore de Huffman

Etapas do algoritmo de Huffman

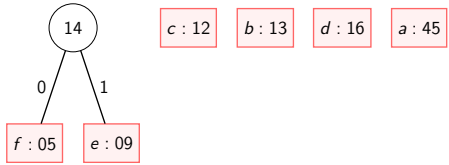
1. Remova da lista de nós, os dois nós menos frequentes
2. Crie um novo nó, cuja frequência seja a soma das frequências dos dois nós retirados
3. Defina como o filho da esquerda desse novo nó, o nó com a menor frequência dos retirados, e como filho da direita o mais frequente
4. Insira esse novo nó na lista ordenada de nós
5. Repita os passos 1 a 4 até restar apenas um nó na lista
6. Esse nó representa a árvore de Huffman

Conjunto inicial de nós

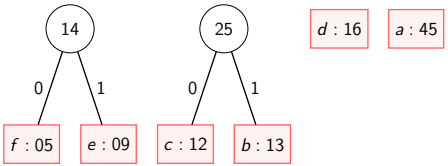
Exemplo: caracteres da tabela 1



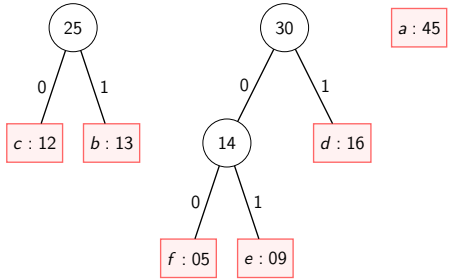
Etapas de montagem da Árvore



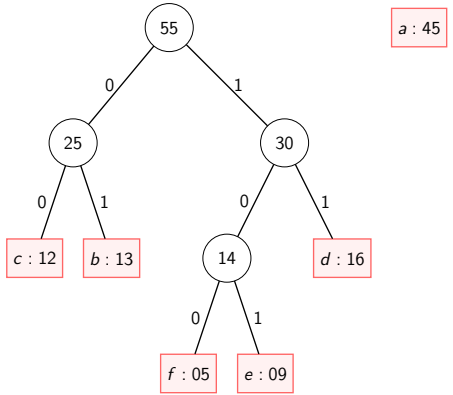
Etapas de montagem da Árvore



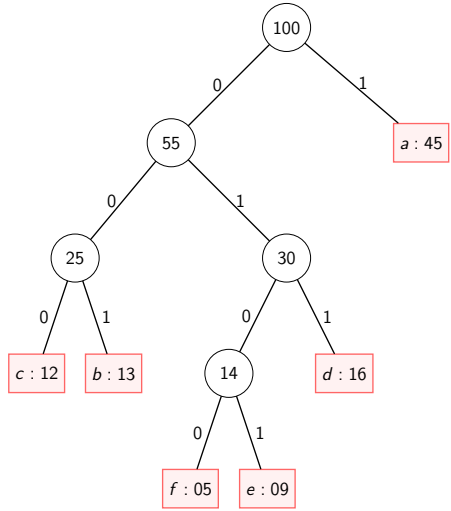
Etapas de montagem da Árvore



Etapas de montagem da Árvore



Árvore Final



Decodificação

- O processo de decodificação do código de Huffman para cada caractere distinto é similar a decodificação de uma mensagem
- Dado um nó folha (caractere), parte-se da raíz até alcançá-lo
  - Se desceu pelo filho da esquerda, acrescenta 0 ao código
  - Se desceu pelo filho da direita, acrescenta 1

	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Codificação	0	101	100	111	1101	1100

Tabela: 3

Codificação variável para cada caractere

Comparativo

- Código de comprimento fixo utiliza 300.000 bits
  - (45x3 + 13x3 + 12x3 + 16x3 + 9x3 + 5x3) x 1.000 = 300.000
- Código de comprimento variável utiliza 224.000 bits
  - (45x1 + 13x3 + 12x3 + 16x3 + 9x4 + 5x4) x 1.000 = 224.000
- Economia de aproximadamente 25% no espaço de representação dos caracteres

Caractere	a	b	c	d	e	f
Frequência (em milhares)	45	13	12	16	9	5
Palavras em comprimento Fixo	000	001	010	011	100	101
Palavras em comprimento Variável	0	101	100	111	1101	1100

Tabela: 4

Comparativo das codificações fixas e variáveis para cada caractere

Códigos de prefixo

- É necessário que a codificação seja não ambígua, de forma que nenhuma palavra formada seja o prefixo de outra palavra
- Tais códigos são chamados de **códigos de prefixo**
- Os códigos de prefixo ajudam a criar compressões ótimas
- Conforme a tabela 4, podemos codificar a palavra abc como
  - $0 \cdot 101 \cdot 100 = 0101100$, onde '$\cdot$' representa a concatenação

Métricas

Complexidade do algoritmo

$\mathcal{O}(nlgn)$ Tempo de execução utilizando com conjunto de n caracteres implementado com heap de mínimo binário

$\mathcal{O}(nlglgn)$ Tempo de execução utilizando com conjunto de n caracteres implementado com árvores de Van Emde Boas

Métricas

Cálculo de número de bits exigidos

T Árvore dada

$c.freq$ A frequência de c no arquivo

$d_T(c)$ Profundidade de folha c na árvore

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c)$$

Código de Huffman

Algoritmo

```
1  Huffman(C)
2   n = |C|
3   Q = C
4   for i=1 to n-1
5           z.esquerda = x = ExtraiMin(Q)
6           z.direita = y = ExtraiMin(Q)
7           z.freq = x.freq + y.freq
8           Insere(Q, z)
9   return ExtraiMin(Q)
```

Código de Huffman

Algoritmo

- Criar uma árvore T de baixo para cima
- Cria-se fila de prioridade mínima Q na linha 3
- Um novo nó com os menores elementos é criado sucessivamente na linha 4-8
- Frequência z é criada pela soma dos nós de menor tamanho na linha 8

Código de Huffman

Prova do Algoritmo

- Seja C um alfabeto que cada elemento c $\in$ C tem a frequência de c.freq
- Considere os elementos x e y em C que contem as menores frequências
  - Existe um código de prefixo ótimo para C cujas palavras de código x e y contem comprimentos iguais com diferença apenas no último bit.
- Queremos provar que uma árvore T que possua um código de prefixo ótimo seja modificada ao ponto de criar uma árvore que represente um outro código de prefixo ótimo, cujo os elementos x e y sejam folhas irmãos com máxima profundidade na árvore.

Código de Huffman

Prova do Algoritmo

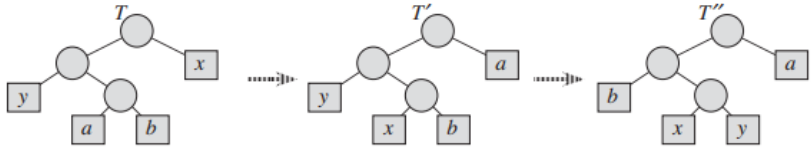


Figura: 2

Prova da corretude do Algoritmo de Huffman

Código de Huffman

Prova do Algoritmo

- Conforme a figura 2 as posições dos elementos a e x foram permutados, criando a árvore T', depois foi permutada a posição b e y em T' para criar a árvore T'' assim permitindo que os elementos a e b sejam irmãos com a maior profundidade.
- Pela equação, podemos provar que a diferença do custo de T e T' é ≥ 0 assim B(T') - B(T'') também é ≥ 0 então B(T'') $\leq$ B(T) sabendo que T é ótimo então B(T) $\leq$ B(T''), o que implica que B(T) = B(T'') então T'' de fato é uma arvore ótima.

Código de Huffman

Diferença ente T e T'

$$
\begin{aligned}
& B(T) - B(T') \\
&= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_{T'}(x) - a.freq \cdot d_{T'}(a) \qquad (1) \\
&= x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_T(a) - a.freq \cdot d_T(x) \\
&= (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\geq 0
\end{aligned}
$$

Utilizações

- Árvores de Huffman são utilizadas em diversas aplicações e formatos
- Dentre os principais e mais conhecidos, podemos citar
  - GZIP
  - PKZIP
  - BZIP2
  - JPEG
  - MPEG
  - PNG

Livros Texto I


Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
Algoritmos teoria e prática.
Elsevier Editora Ltda., 3ª Edição, 2012.


C. H. Papadimitriou, U. V. Vazirani e S. Dasgupta.
Algoritmos.
Mcgraw-Hill Brasil, 2009.


R. Sedgewick, K. Wayne
Algorithms.
Pearson, 4ª Edição, 2014.


M. Cappelle, H. Longo
Estruturas de Dados e Projeto de Algoritmos.