Universidade Federal de Roraima
Departamento de Ciência da Computação
Disciplina: Análise de Algoritmos
Aluno: Josemar Rocha da Silva

Projeto Final A.A. - BST and Heap : Huffman coding and decoding

Primeiramente iremos observar o que é a codificação e decodificação Huffman, para então termos conhecimento para falar como funciona sua implementação.

- **Códigos prefixos:** Na codificação Huffman trabalhamos com sequências de bits, essas que são designadas de uma maneira que um código será designado à um caractere, desta forma o mesmo prefixo de código não será designado a nenhum outro caractere, é desta forma que a codificação Huffman garante que não há ambiguidade quando decodificar a sequência binária que é gerada.

- **Passos para construção de uma codificação Huffman:** Construir uma árvore Huffman com caracteres inseridos; Passar pela árvore designando códigos para cada caractere.

- **Construindo uma árvore Huffman:** Para construir uma árvore será preciso criar um nodo folha para cada caractere único e construir uma heap mínima de cada nodo folha, aqui a heap mínima é usada como fila de prioridade e por fim valor de frequência, este é utilizado para comparar dois nodos na heap mínima. Inicialmente, o caractere menos frequente se encontra na raiz da árvore;

Extraímos os dois nodos com menor frequência da heap mínima e criamos um novo nodo interno que contém a soma da frequência dos dois nodos, o primeiro nodo que foi extraído então se torna o filho esquerda e o segundo o filho direito deste novo nodo interno; Isso se repete até que sobre um nodo, o raiz e então a árvore fica completa.

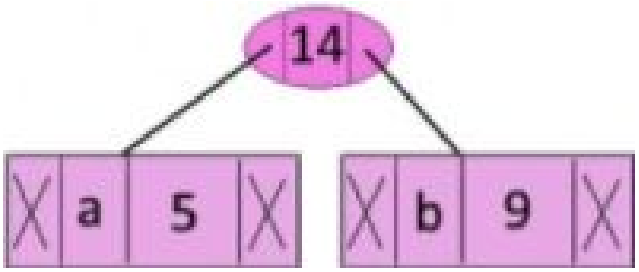
Para concretizar utilizaremos um exemplo:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Dada a seguinte tabela, contendo certos caracteres seguidos de suas respectivas frequências, iremos examinar como a árvore Huffman pode ser criada a partir dela.

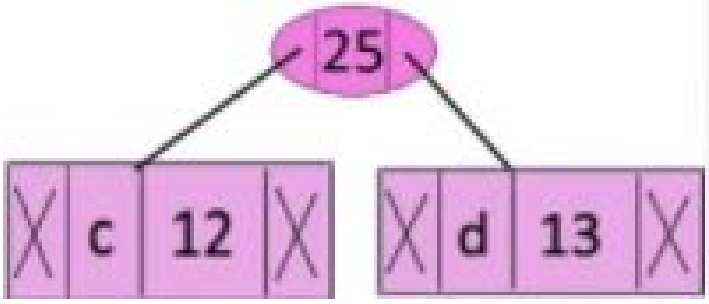
- **Passo 1:** Construímos uma heap mínima contendo 6 nodos onde cada nodo representa a raiz de uma árvore com um único nodo.

- **Passo 2:** Extraímos os dois nodos com menor frequência da heap mínima e adicionamos um novo nodo interno com a frequência 5 + 9 = 14.



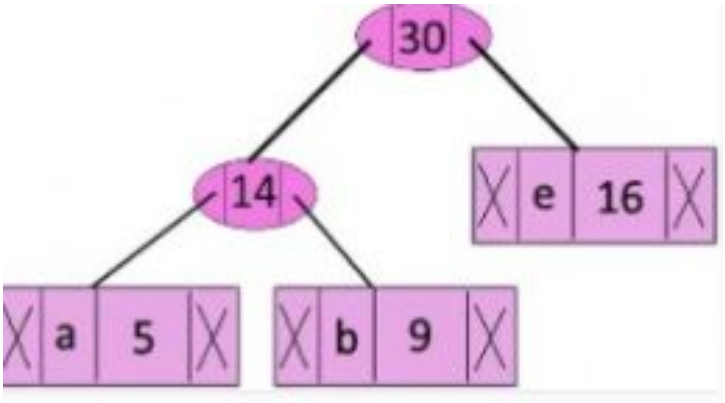
Assim a heap mínima agora contém cinco nodos onde quatro deles são raízes de árvores com um elemento cada e um nodo heap onde a raiz de uma árvore com três elementos.

- **Passo 3:** Repetimos o mesmo processo, e assim adicionamos um novo nodo com frequência 12 + 13 = 25.

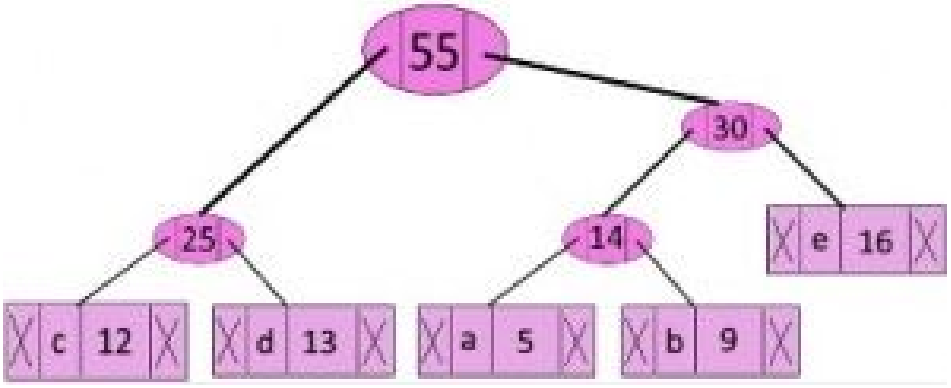


Assim a heap mínima agora contém quatro nodos onde dois deles são raízes de árvores com um elemento cada e dois nodos heap onde a raiz de uma árvore com mais de um nodo.

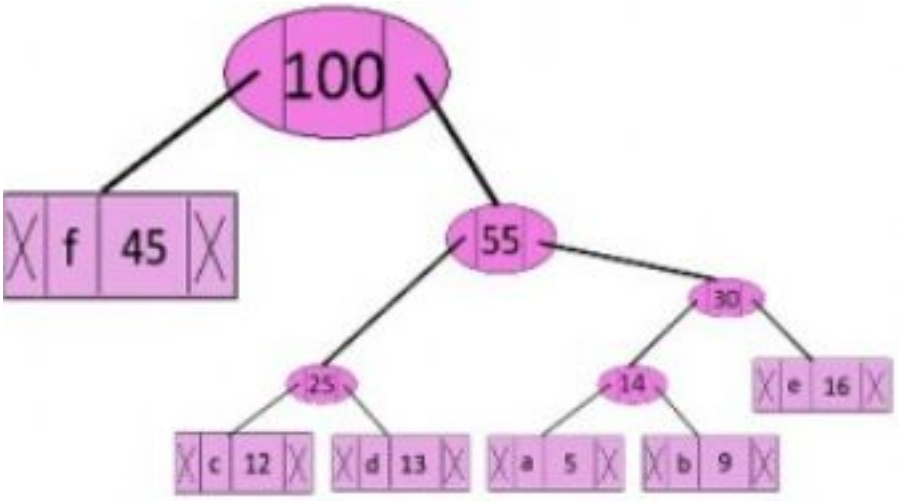
- **Passo 4:** Repetimos o mesmo processo, e assim adicionamos um novo nodo com frequência 14 + 16 = 30.



- **Passo 5:** A partir daqui a heap mínima contém 3 nodos, assim extraímos a frequência do nodos com as menores frequências e adicionamos um novo nodo interno, 25 + 30 = 55.



- **Passo 6:** Agora que a heap mínima contém apenas dois nodos, extraímos então eles e adicionamos um novo nodo com frequência 45 + 55 = 100.



Agora a heap mínima contém apenas um nodo, e o algoritmo para.

Com a árvore Huffman gerada, podemos então navegar pela árvore começando pela raiz mantendo, por exemplo, um vetor auxiliar que seria utilizado para registrar o caminho até cada nodo folha. Quando movermos para o filho à esquerda, colocamos 0 no vetor, quando movemos para o filho à direita colocamos 1 no vetor, quando chegarmos à um nodo folha, podemos imprimir o vetor, ou guardar a informação, por exemplo, em um arquivo onde a informação pode ficar guardado para ser decodificada em um momento futuro por um decodificador.

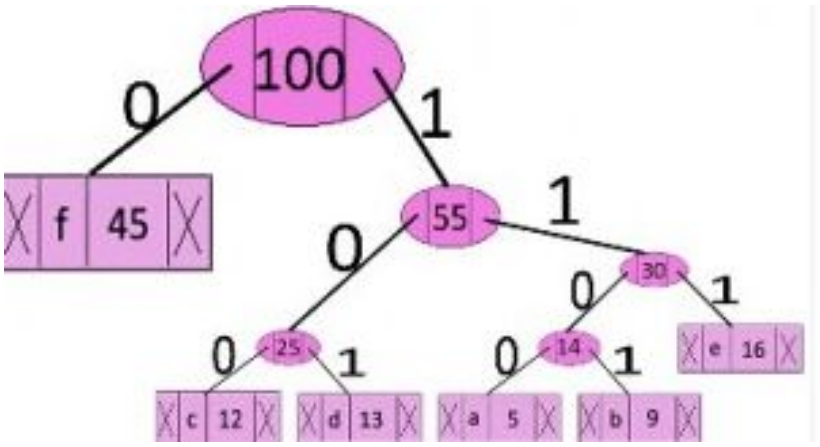


Figura demonstrando como pode ser uma árvore Huffman com seu trajeto até cada nodo folha.

Então assim podemos gerar uma nova tabela com cada caractere e seu respectivo código Huffman.

character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111

- **A decodificação Huffman:** Para decodificar o dados codificados nós precisamos de uma árvore Huffman e iteramos pelo dado binário codificado para achar o caractere correspondente aos bits.

  Primeiramente começamos da raiz e continuamos até que um nodo folha seja encontrado, caso o bit atual seja 0, movemos para o nodo filho à esquerda, caso contrário, nos movemos para à direita, seguimos este processo até que um nodo folha seja encontrado e por fim imprimos ou armazenamos(em um vetor por exemplo) o caracter presente no nodo folha.

- **Complexidade de tempo:** O(n log n), onde n é o número de caracteres únicos; Se existem n nodos, a função para extrair um nodo da heap mínima será chamada 2*(n-1) vezes, esta função leva um tempo de O(log n), pois ela chama uma outra função chamada Heapify. Assim, o tempo em geral, tem uma complexidade de O(n log n).

Referências

<https://www.codeproject.com/Questions/589532/Howplustoplusinputplusaplusfileplusforplus Huffmanp>
<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>
<https://www.geeksforgeeks.org/huffman-decoding/>
<https://www.geeksforgeeks.org/huffman-coding-using-priority-queue/?ref=rp>
<https://www.cs.utexas.edu/users/djimenez/utsa/cs1723/assign7/huffmanc.txt>
<https://www.tutorialspoint.com/huffman-coding>
<https://stackoverflow.com/questions/11587044/how-can-i-create-a-tree-for-huffman-encoding-and-decoding>
<https://rosettacode.org/wiki/Huffman_coding>
<https://github.com/nating/compressor>
<https://www.geeksforgeeks.org/priority-queue-using-binary-heap/>
<https://stackoverflow.com/questions/41666579/huffman-encoding-priority-queue>