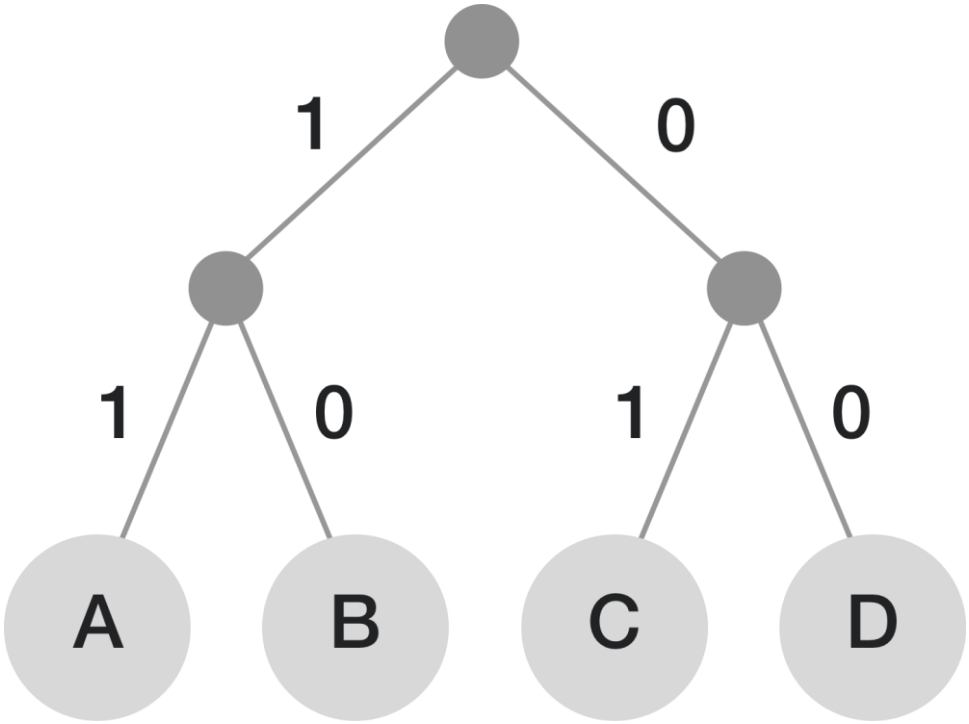


Trabalho Pratico 3

Lucas de Araújo e Lucas Santos de Sá

Ciência da Computação

Amanda Nascimento

24/11/2019

Codificação de Huffman:

A codificação de Huffman é um **algoritmo de compressão sem perdas**, seu princípio básico consiste em criar um formato binário, onde é atribuído menos bits para os símbolos mais frequentes e mais bits para os símbolos menos frequentes.
Dessa forma o tamanho do final arquivo é reduzido.
Foi criado por David A. Huffman durante seu doutorado no MIT e publicado em 1952.

Como funciona:

1) Contar a frequência dos símbolos *(Letras em uma frase por exemplo)*

2) Montar uma árvore binária, agrupando os símbolos por sua frequência de aparição

3) Percorrer a árvore para montar o dicionário binário que será usado como novo código para cada símbolo

4) Recodificar os dados usando este dicionário

Exemplificação:

- Vamos supor que iremos codificar a seguinte frase:
  Ana ama sua nana, sua mana e banana

- Se contarmos cada letra da frase e montarmos uma tabela de frequência, teríamos algo como:

Símbolos	**Frequência**
A	1
n	6
a	12
“espaço”	7
m	2
s	2
u	2
,	1
e	1
b	1

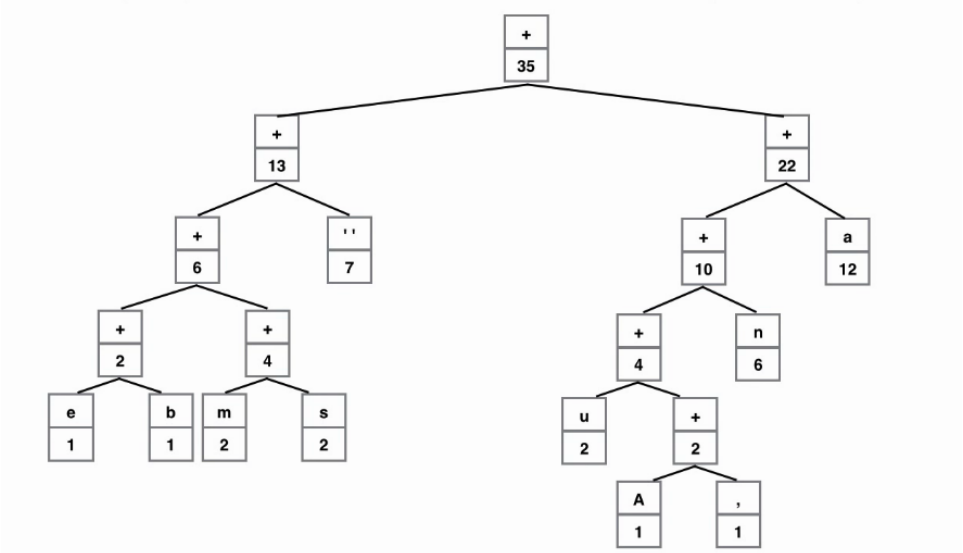
- Agora devemos implementar nossa árvore binária para futuramente utilizarmos nossa codificação
- Para montar nossa árvore, deveremos seguir alguns passos:

  1) *Retiramos os dois nós de menor frequência da lista*
  2) *Agrupamos esses dois nós em um nó pai , deixando o nó de menor frequência a esquerda e a frequência do nó pai será a soma de frequência de seus filhos*
  3) *Adicionamos o nó pai a lista*

- O processo irá acabar quando ao retirarmos os dois nós filhos não houver mais nós na lista

- Nesse processo, os nós contendo letras sempre serão as folhas da árvore

- No final do processo obteremos algo como:



- Os **símbolos mais frequentes ficaram em níveis mais altos** da árvore e os **menos frequentes em níveis mais baixos**
- Com a árvore em mão agora iremos percorrer a árvore e montar uma nova tabela de acordo com as seguintes regras:

  1) *Percorremos a árvore em profundidade e adicionando o bit zero quando formos a esquerda e o bit 1 quando formos à direita*
  2) *Ao encontramos uma folha, adicionamos a nossa tabela*

- No final, obteremos a seguinte tabela:

Símbolo	**Código**
e	0000
b	0001
m	0010
s	0011
“espaço”	01
u	1000
a	10010
,	10011
n	101
a	11

- Agora iremos para a parte final, a codificação em si

- Por exemplo, a palavra *banana* será codificada como:
  0001 11 101 11 101 11
- Dessa forma reduzimos 48 bits para apenas 16 bits e com isso ocupando cerca de 67,37% a menos de memória!

- O processo de decodificação se dá de forma análoga:
  1) *Iniciamos na raiz da árvore*
  2) *Caso o bit seja 0 , percorremos a árvore para a esquerda , caso seja 1 , para direita*
  3) *Ao encontrar a folha, imprimimos o símbolo correspondente*
  4) *Repetimos até que não haja mais bits*

- Por fim, algumas considerações sobre este algoritmo:
  1) *O código de Huffman gera a melhor codificação possível para sequência de símbolos*
  2) *No seu pior caso (sequência uniformemente distribuída), os dados finais ocuparão o mesmo espaço que os originais*
  3) *Para fazer um compactador, será necessário também salvar a árvore e isso pode ser feito de maneira bastante compacta caso a codificação seja convertida para a forma canônica*

Tabela Hash (Parte B)

- O problema abordado é um preenchimento de matriculas de alunos, junto as suas notas e nomes
- Para não correr risco de colisões, foi usado inserção por endereçamento aberto através de segmentação linear
- O programa registra 3 alunos, pesquisa 1 aluno e imprime sua ficha e logo após libera a memória
- Implementações estão no .cpp e .hpp

Complexidade de Tempo	**Complexidade de Memória**
Melhor Caso: O(1)	Melhor Caso: O(N)
Pior Caso: O(N)	Pior Caso: O(N)
Caso Médio: O(1)	Caso Médio: O(N)