BST and Heap : Huffman coding and decoding

•••

Cristiano Lima Sousa Rosa

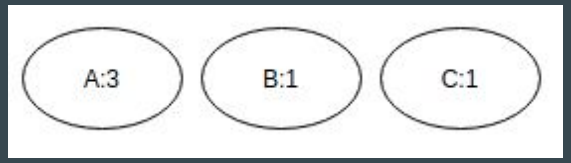
Formas de comunicação

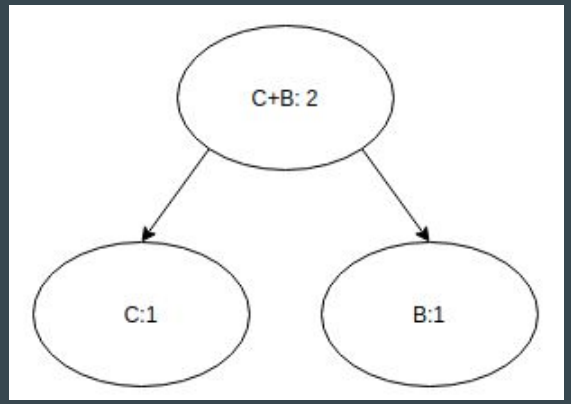
Codificação de Huffman

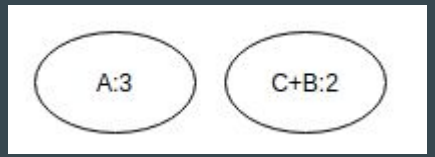
- O Algoritmo precisa das recorrências de caracteres para construir uma árvore

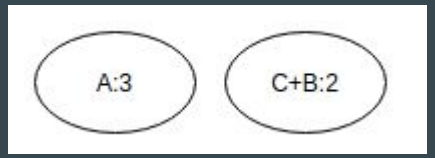
AAABC

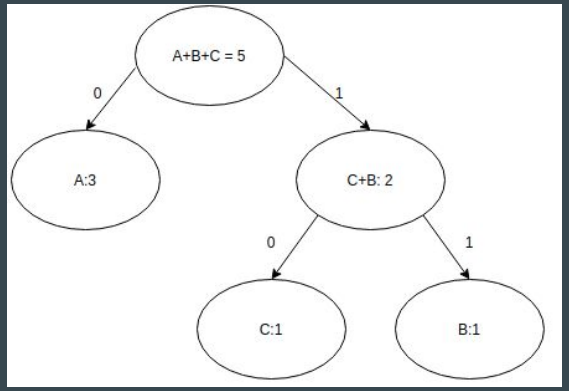
Letra	Frequência
A	3
B	1
C	1


A:3
B:1
C:1


C+B: 2
C:1
B:1

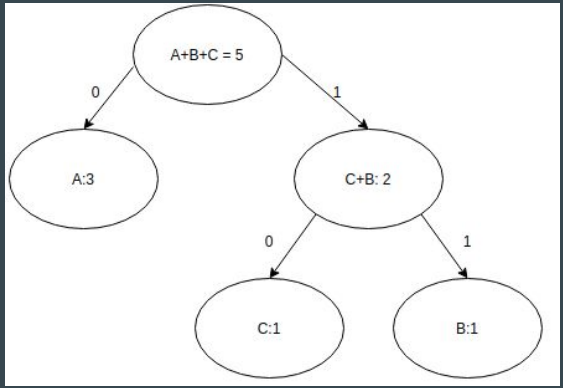

A:3
C+B:2





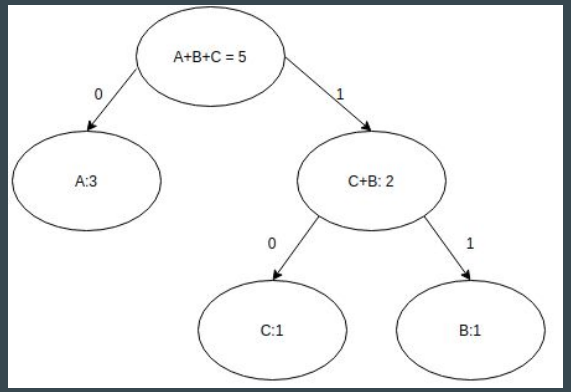
Letra	Codificação
A	0
B	11
C	10

C	C	B	B	CCBB
10	10	11	11	10101111



Letra	Codificação
A	0
B	11
C	10

C	C	B	B	CCBB
10	10	11	11	10101111



Letra	Codificação
A	0
B	11
C	10

C	C	B	B	CCBB
10	10	11	11	10101111

A	65	0100 0001
B	66	0100 0010
C	67	0100 0011

4x 8 bits = 32 bits
CCBB com a codificação terá 8 bits

Implementação

```c
unsigned int pegandoFrequencias(FILE* f, unsigned int vetor[]){
    int r,i;
    for(i = 0;;i++){
        r = fgetc(f);
        if(feof(f)) break;
        vetor[r]++;
    }
    return i;
}
```

Para pegar as recorrências é necessário percorrer N elementos até o final do arquivo
Na inserção os elementos são adicionados na sua devida posição O(1).

```c
node* criarHuffman(int vetorFrequencia[]){
    filaP* f = iniciaFila();
    for(int i = 0; i < N_ASCII;i++){
            if(vetorFrequencia[i]){
                node* auxNode = criarNode(i,vetorFrequencia[i]);
                adicionarNaFila(f,auxNode);
            }
    }
    node* x,*y,*z;
    int n = f->tamanhoHeap-1;
    for(int i = 0; i < n ;  i++){
        z = malloc(sizeof(node));
        x = extraindoMinFila(f);
        y = extraindoMinFila(f);
        z->esq = x;
        z->dir = y;
        z->frequencia = x->frequencia + y->frequencia;
        adicionarNaFila(f,z);
    }
    return extraindoMinFila(f);
}
```

```c
void adicionarNaFila(filaP* f,node* root){
    f->tamanhoHeap++;
    int pos = f->tamanhoHeap-1;

    while ((pos > 0) && (f->vetor[parent(pos)]->frequencia > root->frequencia)) {
        f->vetor[pos] = f->vetor[parent(pos)];
        pos = parent(pos);
    }
    f->vetor[pos] = root;
}
```

```
node* extraindoMinFila(filaP *f){
    node* aux;
    aux = f->vetor[0];
    f->vetor[0] = f->vetor[f->tamanhoHeap-1];
    f->tamanhoHeap--;

    MinHeapify(f,0);
    return aux;
}
```

```c
void MinHeapify(filaP* f,int pos){
    int esq = left(pos);
    int dir = rigth(pos);
    int menor = pos;

    if(esq < f->tamanhoHeap && f->vetor[esq]->frequencia < f->vetor[pos]->frequencia)
        menor = esq;

    if(dir < f->tamanhoHeap && f->vetor[dir]->frequencia < f->vetor[menor]->frequencia)
        menor = dir;

    node* aux;
    if(menor != pos){
        aux = f->vetor[pos];
        f->vetor[pos] = f->vetor[menor];
        f->vetor[menor] = aux;
        MinHeapify(f,menor);
    }
}
```

Complexidade de tempo

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       alocar um novo nó z
5       z.esquerda = x = EXTRACT-MIN(Q)
6       z.direita = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)          // retorna a raiz da árvore.
```

- Operações básicas na Heap são executadas em um tempo que é proporcional a altura da árvore sendo O(logn)
- O loop nas linhas 3 e 8 são executados n-1, como cada operação na heap leva O(logn) e são n elementos no loop a execução do huffman tem o tempo de O(nlogn);

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Algoritmos: Teoria e Prática. 3a edição. Elsevier, 2012. ISBN 9788535236996
- Universo Discreto. Como a compressão de dados funciona? (Árvore de Huffman). Youtube, 12 de fev. de 2021. Disponível em: <https://youtu.be/-TonlL3vcGk>. Acesso em: 11 de out. de 2021.
- Vinícius Godoy de Mendonça.Codificação de Huffman.Youtube,11 de out. de 2015. Disponível em: <https://youtu.be/xQQt5myz00o>. Acesso em: 11 de out. de 2021.