Trabalho 05 - Código de Huffman

Data de entrega: 06/12/2020

Importante:

- Não olhe códigos de outros grupos ou da internet. Exceto o que é fornecido.
- TODOS os membros do grupo devem participar e compreender completamente a implementação.
- Em caso de plágio, fraude ou tentativa de burlar o sistema será aplicado nota 0 na disciplina aos envolvidos.
- Alguns alunos podem ser solicitados para explicar com detalhes a implementação.
- Passar em todos os testes do run.codes não é garantia de tirar a nota máxima. Sua nota ainda depende do cumprimento das especificações do trabalho, qualidade do código, clareza dos comentários, boas práticas de programação e entendimento da matéria demonstrada em possível reunião.
- Você deverá submeter, até a data de entrega, o seu código na plataforma run.codes, onde o "Número de Matrícula" deverá ser o número do seu grupo.

Esse trabalho deverá ser realizado em grupo, com os grupos já definidos na disciplina. Neste trabalho o seu grupo deverá encontrar uma compressão ótima para arquivos, usando o Algoritmo de Huffman.

Normalmente, os caracteres de um arquivo de texto são representados por códigos binários de tamanho fixo, tipicamente 1 byte (8 bits). Na codificação ascii por exemplo, os caracteres, 'a', 'b' e 'n' tem a seguinte codificação:

a - 01100001
b - 01100010
n - 01101110

Dessa forma para escrever a palavra banana, é preciso armazenar 24 bits:

011000100110000101101110011000010110111001100001

Mas se 'a', 'b' e 'n' são as únicas letras utilizadas, conforme visto em aula, poderíamos utilizar um código de tamanho variável livre de prefixo. Por exemplo:

a - 0
b - 11
n - 10

Dessa forma, a palavra banana poderia ser escrita com apenas 9 bits:

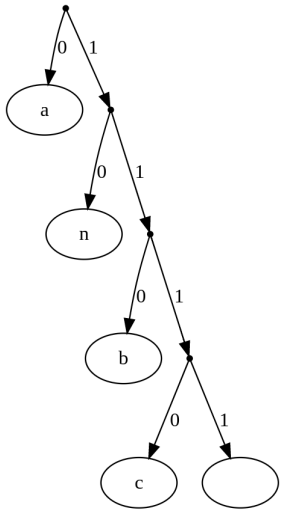
110100100

O Algoritmo de Huffman é um algoritmo Guloso capaz de, dada as frequências com que cada caractere aparece, encontrar a codificação ótima, ou seja aquela que ocupará o menor número possível de bits.

Seu programa deverá ler da entrada padrão do sistema uma entrada textual contendo caracteres que estão compreendidos entre os número 32 e 126 da tabela ascii, um exemplo de uma entrada seria:

```
a banana bacana
```

A entrada acima contém, 2 espaços, 7 ‘a’s, 2 ‘b’s, 1 ‘c’ e 3’n’s. Uma possível árvore criada pelo algoritmo seria essa:



Nessa representação o código de cada caracteres é:

espaço - 1111
a - 0
b - 110
c - 1110
n - 10

De forma que a representação da frase “a banana bacana” nessa codificação seria: “01111110010010011111110011100100”. Note que podem existir codificações diferentes que também podem representar os mesmos caracteres (por exemplo trocando todos os 1s por 0s e vice versa). Dessa forma, você deverá imprimir apenas o número de bits necessários para essa representação. No caso do exemplo considerado, são 31 bits, e seu programa deverá imprimir apenas:

```
31
```

Você receberá uma implementação parcial do algoritmo, cujo o uso é opcional, que já contêm o código para Fila de Prioridade, leitura de entrada e uma forma de converter a árvore criada em uma codificação. Você pode usar e modificar tudo neste código, porém para poupar trabalho apenas

o que está na função main marcado com “TODO” seria necessário modificar. No código fornecido é criado uma árvore qualquer (não ótima) que resulta em uma representação de 34 bits no exemplo acima, e portanto errada.

- Você pode implementar em qualquer linguagem aceita pelo run.codes.
- Seu programa deve executar no run.codes em poucos segundo.
- Se você não tiver certeza se alguma coisa é permitida ou não no trabalho, não hesite em perguntar ao professor!