USP - Universidade de São Paulo
Instituto de Ciências Matemáticas e Computação

Laboratório de Introdução à Ciência da Computação II

SCC0220 - 2021/2

TERCEIRA IMPLEMENTAÇÃO PRÁTICA

CODIFICAÇÃO DE HUFFMAN

Objetivo da implementação: Estimular a prática e o trabalho com estruturas de dados primitivas no âmbito de eficiência. Gerar novas estruturas de dados a partir de estruturas menos elaboradas. Entendimento e implementação de algoritmos a partir de instruções dadas.

1 Introdução

Em computação, sempre houve alta requisição de algoritmos que pudessem realizar a compressão de arquivos (sejam eles imagens, vídeos, músicas e tantos outros) sem a perda de informações. Ao realizarmos a **compressão** de dados, estamos armazenando toda a informação contida nesse conjunto de dados em um subconjunto menor, de tal forma que não possuamos perda de informação entre um processo e outro. Esse último fator é muito importante, visto que recuperaremos o conjunto original de dados a partir do processo de **descompressão** e, possuindo perda de informação, não conseguiríamos obtê-lo novamente.

Em 1952, um estudante de doutorado do MIT chamado David A. Huffman desenvolveu um inovador método de compressão, que permitia eliminar boa parte das redundâncias de informação na forma de *bits*. Seu método de compressão, nomeado **compressão de Huffman**, é um método que leva em consideração a frequência de aparecimento das informações em um conjunto de dados, e possui como principal heurística a de que **"informações mais frequentes devem ser representadas por um número mínimo possível de *bits* após a compressão"**. Além disso, havia a preocupação de que a informação codificada não fosse ambígua, uma vez que esta idealmente seria descomprimida futuramente.

Nesta implementação, **você deverá desenvolver um algoritmo que realize o método de compressão e descompressão de Huffman** para uma cadeia de caracteres dada. A compressão faz mais sentido a nível de *bits*, mas estaremos trabalhando com caracteres (isto é, representando cada bit por um caractere `'0'` ou `'1'`) para facilitar a visualização e explicitar melhor seu funcionamento. Acompanhe os próximos parágrafos para uma descrição detalhada do algoritmo e de como construí-lo, utilizando de estruturas de dados que já conhecemos.

2 Noções básicas

Antes de desenvolvermos o algoritmo diretamente, precisamos entender seu embasamento teórico para definir que tipo de informações precisamos extrair da cadeia de caracteres.

A ideia central no algoritmo de Huffman é utilizar da vantagem de que "informações com grande frequência no conjunto de dados devem ser representadas pelo número mínimo possível de daquele momento". Assim, torna-se necessário que primeiro entremos em contato com a ideia de alfabetos e frequências.

2.1 Alfabetos e frequências

Dado um conjunto finito A, com n pares ordenados da forma $A = \{(\alpha_1, w_1),\ (\alpha_2, w_2), \ldots, (\alpha_n, w_n)\}$ e gerado a partir do conjunto de dados dado de entrada, chamaremos-o de **alfabeto**. Além disso, denominaremos cada par ordenado $(\alpha_i, w_i),\ i \in \{1, 2, \ldots, n\}$ de **símbolo desse alfabeto**, onde α_i denota a **representação visual desse simbolo** no conjunto de dados e w_i diz respeito ao seu **peso, ou frequência**, relativo ao conjunto original de dados.

- Consideremos, por exemplo, que estamos trabalhando no conjunto de dados que compõe a palavra "ICMC", no alfabeto tradicional. O conjunto alfabeto A que descreve essa palavra será, intuitivamente, $A = \{(\mathrm{I}, 1),\ (\mathrm{C}, 2),\ (\mathrm{M}, 1)\}$.

Com esse embasamento bem definido, podemos agora ir de encontro ao algoritmo de Huffman, nosso objetivo inicial.

2.2 Algoritmo de Huffman

A ideia intuitiva do algoritmo de Huffman para codificação de dados é receber um conjunto de dados inicial I_1 (nesse caso, trabalharemos com *strings*) e devolver outro conjunto de dados I_2, contendo todas as informações do conjunto inicial (isto é o mesmo que dizer que podemos obter I_1 a partir de I_2 ao aplicarmos o processo de **descompressão**) e de maneira que o peso total de I_2 seja menor do que I_1.

- Como aqui estamos trabalhando com *strings*, sabemos que cada caractere convencional desta é representado, usualmente, por um conjunto de 8 *bits* de informação. Não seria uma boa ideia, então, criarmos uma relação entre sequências binárias menores de 8 *bits* e os caracteres inicias, de maneira com que a cadeia final de *bits*, após a compressão, se torne menor do que a original?

- Um breve pensamento pode nos dizer que, a princípio, é melhor que deixemos os caracteres com mais frequência (isto é, que irão se repetir mais vezes) com as representações binárias de menor tamanho. Por exemplo, se em uma *string* o caractere "`a`"se repete 8 vezes e todos os restantes apenas duas vezes, a intuição nos diz que é preferível que admitamos a representação binária mais curta, "`0`", para o caractere "`a`"já que este é o que mais se repete na cadeia de caracteres.

- Precisamos considerar também que cada caractere c, antes da codificação, deve possuir um representante codificado $\bar{c}$ único. Isto é, ao ser aplicada a descompressão, $\bar{c}$ deverá resultar unequivocamente em c, sem ambiguidade.

A geração de I_2, felizmente, pode ser feita atendendo todas as condições acima através de uma estrutura de dados denominada **árvore de Huffman (ou trie[1] de Huffman)**, que é construída através de um conjunto alfabeto A originado do conjunto de dados de entrada e possui sua base na estrutura de **árvore binária**.

2.3 Árvore de Huffman

Essa estrutura de dados possui um método de geração bem definido, e depende diretamente do conjunto de dados que será informado de entrada. Assim, podemos descrever esse processo como:

1. A partir do conjunto inicial de dados, gere o conjunto alfabeto A equivalente. Isso é o mesmo que, em outras palavras, analisar a frequência de cada caractere na *string* e formar um conjunto com esses dados, como especificado na seção 2.1.
2. Para cada símbolo $(\alpha_i, w_i),\ i \in \{1, 2, \ldots, n\}$, do conjunto alfabeto A, crie um **nó folha de árvore binária** (isto é, um nó que pode possuir outros dois filhos) que armazene a representação visual (α_i) e a frequência (w_i) desse símbolo.
3. Insira cada nó criado no passo 2 em um conjunto de nós.
4. Do conjunto, retire **sem reposição** dois nós p_1 e p_2 relativos aos nós que armazenam a menor frequência dentre os inseridos no conjunto.
5. Crie um novo nó de árvore binária p_3, cuja frequência será determinada pela soma das frequências dos símbolos de p_1 e p_2. A representação visual nesse nó é desprezível e pode ser deixada como vazia.
6. Faça com que p_1 e p_2 se tornem filhos de p_3. Dentre os dois, o nó que armazena a menor frequência deve ser filho esquerdo de p_3, enquanto o outro deve ser filho direito.
7. Insira o nó criado p_3 no conjunto.
8. Volte ao passo 4 e repita o processo, até que reste apenas um nó no conjunto. Esse nó será o nó raiz da nossa árvore de Huffman, que possuirá ligação direta com todos os outros.
9. Retorne o nó raiz que restou no conjunto.

A estrutura de dados gerada é a chave do que nos possibilitará aplicar a codificação/decodificação de cada caractere independentemente.

[1] Uma trie é uma estrutura de dados particular derivada das árvores com um funcionamento muito interessante. Pode ser muito útil na resolução de muitos problemas. Vale a pesquisa!

2.4 O processo de compressão

A árvore de Huffman é a ferramenta fundamental para que a compressão possa ocorrer. A partir do nó raiz da árvore, o processo de compressão é feito através de um **percurso pós-ordem** com uma heurística bem definida: considere que a representação codificada de um caractere c, enquanto estamos no nó raiz, é uma cadeia de caracteres S inicialmente vazia. A partir disso,

- Ao ir por um nó **a esquerda**, acrescente o caractere `'0'` no final de S.
- Ao ir por um nó **a direita**, acrescente o caractere `'1'` no final de S.
- O processo se repete até que o nó folha de c seja atingido. Quando isso ocorrer, a string S representa a forma codificada de c.

O mesmo processo pode ser repetido para todos os nós folhas da árvore, assim obtendo a forma codificada de todos os caracteres inicialmente processados. Com isso, é possível codificar a mensagem dada de entrada.

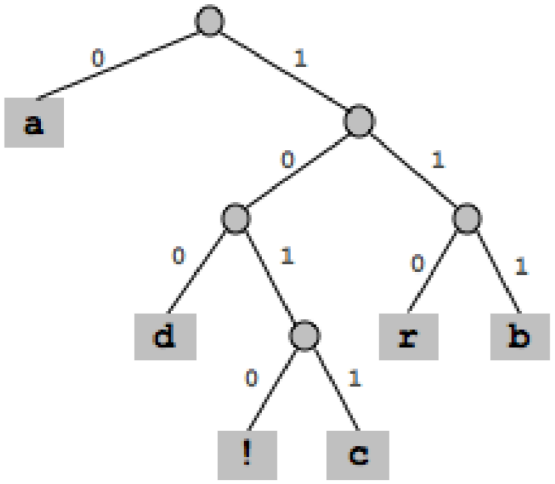
Nota: A string S codificada pode variar dependendo da ordem que dois nós de igual peso forem retirados do conjunto. No entanto, a taxa de compressão (isto é, o percentual comprimido) é invariável caso seja feita uma solução ótima. Entenda por "solução ótima"a melhor solução possível para o problema apresentado.

2.5 O processo de descompressão

A partir da árvore de Huffman construída, é possível realizar a descompressão com base na cadeia binária S (no nosso caso, de caracteres `'0'` e `'1'`) gerada pela compressão. Intuitivamente, o modelo é bem simples e segue os mesmos passos do anterior:

- Comece percorrendo, a partir nó raiz da árvore de Huffman, os caracteres contidos na cadeia S. A lógica de percurso da árvore se mantém a mesma da compressão: no caso do caractere atual de S ser `'0'`, vá para o nó a **esquerda**; caso contrário (seja `'1'`), vá para o nó a **direita**.
- Repita o processo até que um nó folha seja atingido. Quando isso ocorrer, significa que um caractere codificado pôde ser decodificado. O algoritmo deve, então, retornar ao nó raiz da árvore e continuar com o mesmo processo, a partir da posição corrente de S, até que a cadeia chegue ao fim.

A imagem a seguir (Figura 1, próxima página) possui uma representação visual de uma árvore de Huffman, que permite que sejam gerados os caracteres do conjunto $C = \{A, B, C, D\}$. Note que a mesma árvore utilizada para codificar a mensagem, também é utilizada para decodificá-la.



char	encoding
a	0
b	111
c	1011
d	100
r	110
!	1010

Figura 1: Árvore de Huffman para o conjunto C, expressa como uma árvore binária.

3 Informações adicionais

- A complexidade de tempo do algoritmo de codificação de Huffman deve ser $\mathcal{O}(n \log(n))$ — para n nós da árvore de Huffman, a função de busca e resgate do valor mínimo contido na lista deve ser feita em tempo $\mathcal{O}(\log(n))$. Qual estrutura de dados permite rapidamente resgatar o valor mínimo dentre um conjunto de dados de maneira eficiente? Considere que realizar a ordenação não é uma opção, visto que não existe ordenação $\mathcal{O}(\log(n))$.
- A complexidade de espaço do algoritmo de codificação de Huffman deve ser de, no máximo, $\mathcal{O}(n)$ para n nós da árvore.
- As estruturas de dados utilizadas ao longo do processo devem ser implementadas pelo próprio autor. Considere separar em diversos arquivos, modularizando coerentemente, e realizar a entrega fazendo uso do Makefile.

4 Entrada

De entrada, será dada uma cadeia de caracteres de tamanho variável que deverá ser utilizada para gerar a árvore de Huffman e, consequentemente, comprimida a partir dela. A *string* será informada pela entrada padrão.

- A cadeia de caracteres pode possuir espaços e pontuações, e esses devem ser considerados na compressão.

- Todos os caracteres estão presentes na tabela ASCII.
- O caractere `'\n'` não deve ser considerado na construção da árvore. Todas as entradas não terão mais do que uma linha.

5 Saída

Na saída padrão, devem ser exibidas duas linhas: na primeira, encontra-se a mesma cadeia de caracteres dada de entrada, que deve ser originada a partir da cadeia de caracteres S codificada durante a execução do programa. Na segunda linha, deverá haver o índice de compressão I_{cp}, $0 \leq I_{cp} \leq 1$, acompanhando do rótulo `'Indice de compressao: '`.

- O índice de compressão é um real com precisão dupla (`double`), que segue o modelo de arredondamento padronizado (a máscara da função `printf` já cuida disso para você!).
- A partir da mensagem dada de entrada, ela deverá ser comprimida e, logo após, descomprimida, resultando na cadeia de caracteres original de entrada. Caso sua cadeia de caracteres descomprimida seja igual a de entrada e o índice de compressão seja o mesmo da saída, você atingiu a solução ótima.
- O índice de compressão pode ser calculado da forma $I_{cp} = \frac{1}{8}\left(\frac{\text{size}\{s_{cp}\}}{\text{size}\{s_0\}}\right)$, onde s_0 representa a cadeia de caracteres original e s_{cp} a cadeia de caracteres após a compressão (em caracteres `'0'` e `'1'`).

6 Exemplos de entrada e saída

Nota: Note que a indentação em algumas linhas representa apenas que a linha abaixo continua na mesma linha (que, por sinal, é potencialmente bem grande). Esse é um propósito plenamente visual e não deve ser copiado nos casos de teste.

- Exemplo 1:

Entrada

```
Ao verme que primeiro roeu as frias carnes do meu cadaver dedico como saudosa lembranca
    estas memorias postumas.
```

Saída

```
Ao verme que primeiro roeu as frias carnes do meu cadaver dedico como saudosa lembranca
    estas memorias postumas.
Indice de compressao: 0.49
```

- Exemplo 2:

Entrada

```
And then one day you find ten years have got behind you. No one told you when to run,
    you missed the starting gun.
```

Saída

```
And then one day you find ten years have got behind you. No one told you when to run,
    you missed the starting gun.
Indice de compressao: 0.50
```

Bom trabalho!