Construção e Análise de Algoritmos

lista de exercícios 18

1. **Alguns casos particulares**

   Considere um alfabeto de n letras associadas às frequências $f_1, f_2, \ldots, f_n$.

   a) Argumente que se alguma frequência f_j é maior ou igual a $2/5$, então um código de prefixo ótimo para esse alfabeto certamente tem alguma letra com código de 1 bit.

   b) Argumente que se todas as frequências são menores que $1/3$, então um código ótimo para esse alfabeto certamente não tem nenhuma letra com código de 1 bit.

   c) Qual é o maior código que uma letra pode ter em um código de prefixo ótimo?
   Em que condições isso pode acontecer?

2. **Frequências balanceadas**

   Suponha que a maior frequência de todas é no máximo 2 vezes maior que a menor frequência de todas

   $$\mathsf{max}_i\, f_i \quad \leq \quad \mathsf{min}_i\, f_i$$

   Qual é a árvore de codificação produzida pelo algoritmo guloso da aula 18, nesse caso?

3. **Códigos sob medida**

   Suponha que alguém gosta muito das letras `o` e `g`.

   Quando essa pessoa vai escrever alguma coisa, ela sempre procura usar palavras que contém essas letras (de preferência as duas) — por exemplo, *gostosura*, *engomadinho*, *amargoso*, *engenhoca*, *foguetório*, *gordurinha*, *agigantado*, *bangalô*, *fedegoso*, e por aí vai ...

   Essa preferência, é claro, vai aumentar a frequência das letras `g` e `o`, mas é possível que isso também afete as frequências das outras letras.

   O ponto é que, se utilizarmos um código baseado na frequência média de ocorrência das letras no português, nós não vamos produzir a menor mensagem possível nesse caso.

   Quer dizer, o ideal seria usar um código sob medida para codificar os textos dessa pessoa.

   Mas, se a moda pega, cada um agora vai querer ter o seu próprio código de prefixo.

   De fato, isso não tem problema nenhum.

   Para fazer a coisa funcionar, basta que a mensagem inclua também uma descrição do código que está sendo utilizado para codificá-la.

   Mostre que $2n - 1 + n \log n$ bits bastam para descrever a árvore de decodificação de um código de prefixo. (Dica: utilize $2n - 1$ bits para descrever a estrutura da árvore, e $n \log n$ bits para descrever a permutação das letras nas folhas.)

4. **Atualização de código**

   Imagine que você construiu um código de prefixo ótimo para um alfabeto com n letras.

   Daí, alguém diz para você: "*Me desculpe, mas eu esqueci de te passar a frequência do* `j`".

   Quer dizer, o seu código está incompleto.

   Agora, você tem basicamente duas alternativas: reconstruir o código do zero, ou atualizar a árvore de decodificação para incorporar o novo símbolo.

   Suponha que você escolhe a segunda opção.

   Descreva uma maneira eficiente de realizar essa tarefa.

   Quando é que isso é uma boa ideia? (i.e., quando a frequência de `j` é grande ou pequena?)

5. **Códigos com pausas** (para pensar)

   Talvez por causa do nome, Samuel simpatizou bastante com a ideia do código Morse, e queria encontrar uma maneira de fazer a coisa funcionar mesmo com as pausas.

   Uma ideia simples seria pensar nas pausas como um terceiro símbolo que seria utilizado para codificar as mensagens (como um "bit"2 adicional, por exemplo).

   Depois de pensar um pouco, Samuel viu que não seria difícil adaptar as ideias da aula 18 para produzir um código de prefixo ótimo que utiliza 3 símbolos para codificar as mensagens.

   (Você consegue ver isso também?)

   Mas, isso não daria a solução que ele queria.

   Quer dizer, se a pausa fosse utilizada como um símbolo adicional, poderia ser o caso de que alguma letra teria uma pausa no meio do seu código, e não era assim que o código Morse funcionava.

   A ideia é que a pausa deveria aparecer no máximo uma vez, e apenas como o último símbolo de um código.

   a) Apresente um algoritmo que constrói um código com pausas a partir das frequências de ocorrência das letras de um alfabeto.

   b) Argumente que o seu algoritmo encontra um código com pausas ótimo.

   c) Compare a eficiência de um código com pausas ótimo com a eficiência de um código de prefixo ótimo.