Solução Lista 02 - Algoritmos Gulosos - CANA 2025.1

Cornélio A. de S.

Maio 2025

Q1

(a)

Desenvolvendo o somatório do tamanho da codificação, temos:

$$\sum_{i=1}^{n} mp_i \log_2 \frac{1}{p_i} = \sum_{i=1}^{n} mp_i \log_2 2^{k_i} = \sum_{i=1}^{n} mp_i k_i$$

Percebe-se que a última equação é a fórmula para o custo de uma codificação de Huffman, onde k_i representa o custo de um símbolo de frequência mp_i, ou seja, sua profundidade na árvore de codificação. Portanto, para provarmos a equação acima, basta mostrar que um símbolo terá profundidade k se e somente se possuir probabilidade $1/2^k$. Faremos essa prova por indução, começando com o caso base que são os símbolos com profundidade 1. Porém, antes disso, vamos provar que, dada as restrições do problema (probabilidades da forma $1/2^{k_i}$), os filhos de um mesmo nó devem sempre ter a mesma probabilidade, pois essa prova nos será útil durante a indução:

Propriedade do parentesco: filhos de um mesmo nó devem ter probabilidades iguais. Consequentemente, cada filho tem metade da probabilidade do pai.

Vamos provar a propriedade acima. Digamos que os símbolos a serem codificados são $\{a_1, a_2, \ldots, a_n\}$ (em ordem crescente de probabilidade) e que o símbolo de menor frequência, a_1, tenha probabilidade $1/2^k$. Demostraremos que, para que a soma das probabilidades de todos os símbolos seja igual a 1, é necessário que o símbolo a_2 também tenha probabilidade $1/2^k$.

Dado que $1/2^k$ é a menor probabilidade dentre os símbolos do alfabeto, podemos ter no máximo 2^k símbolos com probabilidade $1/2^k$, uma vez que $2^k \cdot 1/2^k = 1$. Se o alfabeto tiver menos que 2^k símbolos ($n < 2^k$), podemos somar as probabilidades $1/2^k$ entre si de modo a reduzir o número de probabilidades e manter o somatória igual a 1. Porém, pra obtermos apenas símbolos com probabilidades da forma $1/2^{k_i}$, temos que obrigatoriamente combinar duas probabilidades iguais:

$$\frac{1}{2^j} + \frac{1}{2^j} = 2 \cdot \frac{1}{2^j} = \frac{1}{2^{j-1}}$$

Dentre as 2^k probabilidades iguais à $1/2^k$, pelo menos uma não pode ser somada as outras, uma vez que assumimos que pelo menos o símbolo a_1 tem probabilidade igual à $1/2^k$. Nos resta então $2^k - 1$ probabilidades iguais à $1/2^k$ para serem combinadas entre si duas à duas de modo a obtermos probabilidades da forma $1/2^{k-1}$. Como $2^k - 1$ é ímpar, irá sobrar pelo menos uma

probabilidade $1/2^k$ que não pode ser combinada, fazendo com que pelo menos mais um símbolo tenha probabilidade $1/2^k$, neste caso o símbol a_2 que é o segundo com menor probabilidade dentre os símbolos do alfabeto.

Uma vez que a_2 também tem probabilidade $1/2^k$, o algoritmo formador da árvore mínima da codificação de Huffman irá combinar a_1 e a_2 em um novo símbolo S de probabilidade $P(S) = 1/2^{k-1}$. Se $P(a_3)$ também for igual a $1/2^k$, teremos a mesma situação novamente, o que irá obrigar $P(a_4) = 1/2^k$ e fará com que o algoritmo de Huffman combine a_3 e a_4, e assim por diante até serem esgotados os símbolos de probabilidade $1/2^k$. Uma vez que o algoritmo de Huffman combinar todos os símbolos de probabilidade $1/2^k$, teremos um novo alfabeto $\{a'_1, a'_2, \ldots, a'_{n-p}\}$ (onde p é o número de combinações que foram feitas) cujas probabilidades também são potências de 2 e cuja menor potência corresponde ao símbolo a'_1 com o valor de $1/2^{k-1}$. As mesmas ideias podem ser aplicadas à esse alfabeto, o que obriga a'_2 a também ter probabilidade $1/2^{k-1}$ e faz com que o algoritmo de Huffman combine a'_1 e a'_2. Portanto, por indução, o algoritmo de Huffman irá sempre combinar símbolos de probabilidades iguais, o que significa que todo os nós com filhos terão filhos com a mesma probabilidade. ■

Agora, vamos provar por indução que os símbolos de profundidade k possuem probabilidade $1/2^k$ e vice-versa, começando com o caso base que é $k = 1$.

CASO BASE: Nós (nós internos ou folhas) de profundidade 1 são filhos do nó raiz. Ambos os filhos da raiz devem ter probabilidades iguais (segundo a propriedade do parentesco acima) e uma vez que o nó raiz tem probabilidade 1 (a raiz acumula a probabilidade de todos os símbolos) os seus dois filhos devem ter probabilidade 1/2. Portanto, símbolos de profundidade 1 devem ter probabilidade 1/2. Ademais, não é possível que símbolos com probabilidade 1/2 tenham profundidade maior que 1, uma vez que todo nó abaixo da profundidade 1 deve ter probabilidade menor que 1/2, visto que são descendentes de nós cuja probabilidade é 1/2.

HIPÓTESE INDUTIVA: Para $k \in \{1, 2, \ldots, h\}$ os nós de profundidade k possuem probabilidade $1/2^k$ e vice-versa.

PASSO INDUTIVO: Dada como verdadeira a hipótese indutiva, temos que provar que os símbolos de profundidade $h+1$ possuem probabilidade $1/2^{h+1}$. Para um símbolo ter profundidade $h+1$ ele tem que ser filho de um nó de profundidade h. Nós de profundidade h, pela hipótese indutiva, tem probabilidade $1/2^h$ e seus filhos, segundo a propriedade do parentesco, devem ter probabilidades iguais. Portanto, nós de profundidade $h+1$ devem ter probabilidade $1/2^h \div 2 = 1/2^{h+1}$. Ademais, não é possível que símbolos com probabilidade $1/2^{h+1}$ tenham profundidade maior que $h+1$, uma vez que todo nó abaixo da profundidade $h+1$ deve ter probabilidade menor que $1/2^{h+1}$.

Portanto, segue por indução que símbolos cuja probabilidade são da forma $1/2^{k_i}$ possuem profundidade k na árvore de codificação mínima de Huffman se e somente se tiverem probabilidade $1/2^k$. Essa propriedade é equivalente a equação de tamanho da codificação:

$$\sum_{i=1}^{n} mp_i \log_2 \frac{1}{p_i}$$

(b)

A fórmula é máxima quando a distribuição é uniforme. A fórmula é mínima quando apenas um valor concentra toda a probabilidade, ou seja, tem probabilidade 1 enquanto os outros valores tem prababilidade 0.

Q2

Modificar o algoritmo de Kruskal para obter a Árvore Geradora Máxima em vez da mínima (inverter os pesos ou ordenar em reverso) e retornar as arestas em E que não fazem parte desta árvora. Solução no arquivo */src/cana/feedback.py*.

Q3

(a)

Para um símbolo ter uma codificação de comprimento 1, este símbolo deve "sobreviver" até a última combinação de símbolos que forma a raíz da árvore. Consideremos que os símbolos em ordem crescente de frequência sejam $\{a_1, a_2, \ldots, a_{n-1}, a_n\}$, onde $F(a_i) = f_i$ é a frequência do i-ésimo símbolo a_i e tendo $f_n > 2/5$. Podemos ter no máximo 2 símbolos com frequência maior ou igual a $2/5$, então, se f_{n-1} também for maior (ou igual a) que $2/5$, temos que:

$$\left(\sum_{i=0}^{n-2} f_i\right) < 1/5$$

Dessa forma, o algoritmo de Huffman irá combinar todos os símbolos do índice 1 até o índice $n-2$ em um "novo" símbolo, vamos chamá-lo de S, cuja frequência é $F(S) = \sum_{i=0}^{n-2} f_i < 1/5$. Dessa forma, nos restará 3 símbolos S, a_{n-1} e a_n, com frequências $F(S) < f_{n-1} \leq f_n$. O restante do processo de formação da árvore mínima irá combinar S e a_{n-1} e o resultado será combinado com a_n, fazendo com que a_n tenha uma codificação de comprimento unitário.

Agora vamos cobrir o caso em que $a_{n-1} < 2/5$. Considere o momento em que o processo de combinação de símbolos do algoritmo de Huffman resulte *no primeiro símbolo com frequência maior ou igual a* f_n, de modo que tenhamos agora os símbolos $\{S_1, S_2, \ldots, S_{j-1}, S_j\}$ em ordem crescente de frequência. Como foi a primeira vez que o algoritmo retornou uma combinação de símbolos com frequência maior (ou igual a) que f_n, essa combinação deve ser o símbolo S_j, que é agora o símbolo com a maior frequência dentre as combinações. Ademais, temos que ter obrigatoriamente $S_{j-1} = a_n$, pois se a_n já tiver sido combinado com algum outro símbolo, S_j não seria o primeiro a ter frequência maior que f_n, o que é um absurdo frente a nossa consideração inicial.

Obviamente, se $f_n \geq 1/2$, não é possível que uma combinação de símbolos que não contenha a_n tenha uma frequência maior que f_n, mas nesses casos é óbvio que a codificação de a_n terá comprimento unitário, uma vez que o algoritmo de Huffman iria combinar todos os símbolos $\{a_1, a_2, \ldots, a_{n-1}\}$ para só então combinar o resultado com a_n.

Agora temos que provar por absurdo que os símbolos $\{S_1, S_2, \ldots, S_{j-2}\}$ não podem existir. Temos as seguintes inequações até agora:

$$\begin{aligned} F(S_{j-1}) = F(a_n) = f_n > 2/5 \\ \Rightarrow F(S_j) \geq f_n > 2/5 \\ \Rightarrow F(S_{j-1}) + F(S_j) > 4/5 \\ \Rightarrow \left(\sum_{i=1}^{j-2} F(S_i)\right) < 1/5 \end{aligned}$$

Agora vamos nos questionar sobre os símbolos K_1 e K_2 que formaram S_j:

$$F(K_1) + F(K_2) = F(S_j) \geq f_n > 2/5$$

Para que a frequência de S_j seja maior que $2/5$, um dos símbolos K_1 ou K_2 deve ter uma frequência maior que $(2/5 \div 2) = 1/5$. Porém, isso é um absurdo, pois o algoritmo de Huffman apenas combina os símbolos de menor frequência dentre todos os que estão disponíveis e nós temos os símbolos $\{S_1, S_2, \ldots, S_{j-2}\}$ todos com frequências menores que $1/5$. A única forma de solucionar este absurdo é se $\{S_1, S_2, \ldots, S_{j-2}\}$ não existirem, havendo apenas $\{a_n, S_j\}$. Dessa forma, a_n também terá codificação unitária quando $F(a_{n-1}) = f_{n-1} < 2/5$. ■

(b)

Novamente, para um símbolo ter uma codificação de comprimento 1, este símbolo deve "sobreviver" até a última combinação de símbolos que forma a raíz da árvore. Vamos supor que existe um conjunto de símbolos $\{a_1, a_2, \ldots, a_n\}$ com frequências $\{f_1, f_2, \ldots, f_n\}$ todas menores que $1/3$, ordenados de forma crescente com base nas frequências. Para haver uma codificação de comprimento 1 é necessário que a sequência de combinações de símbolos do algoritmo de Huffman resulte no seguinte conjunto de símbolos:

$$\{a_n, S\} \quad \text{onde} \quad F(a_n) < 1/3 \quad \text{e} \quad F(S) > 2/3$$

Digamos que S foi formado pela combinação dos símbolos K_1 e K_2. Para tal, esses símbolos devem ser os dois símbolos com as menores frequências no conjunto $\{K_1, K_2, a_n\}$. Como $F(S) > 2/3$, um dos símbolos K_1 ou K_2 deve ter frequência maior que $1/3$, o que é um absurdo, pois K_1 e K_2 devem ter frequências menores ou iguais à frequência do símbolo a_n. ■

Q4

Vamos assumir que G é conectado. A estratégia é remover todos os vértices em U do grafo G, aplicar o algoritmo de Kruskal para determinar a árvore geradora mínima neste novo grafo reduzido e depois adicionar os nós removidos usando as arestas incidentes de menor custo para cada um.

Se qualquer um dos vértices em U for um nó essencial para a conectividade do grafo, ou seja, ao remover o nó o grafo que antes era totalmente conectado se torna 2 ou mais componentes conectadas, não é possível que tal nó seja uma folha em qualquer que seja a árvore geradora construída. É possível identificar tais situações analisando o resultado do algorítmo de Kruskal: se o número de aresta retornado por Kruskal for menor que $|V - U| - 1$, então pelo menos um dos vértices removidos era essencial para a conectividade do grafo e o resultado final não é alcançável.

Q5

Basta utilizar a mesma estrutura de dados de árvores enraizadas para conjuntos disjuntos utilizada no algoritmo de Kruskal. Utilizamos a união por rank para unir árvores com base no conjunto de restrições de igualdade. Uma vez feita todas as uniões e esgotado os pares de igualdades, basta verificar a validade das restrições de desigualdade: para uma desigualdade ser

válida, as variáveis da desigualdade devem estar em árvores enraizadas distintas, caso contrário, a desigualdade é inválida e as restrições não são satisfatível.

Q6

Atender os clientes em ordem crescente do tempo para serem servidos. Ou seja, começar com os clientes que podem ser atendidos mais rapidamente. Algoritmos de ordenação eficientes tem complexidade $O(n \log n)$. Se os tempos forem pequenos ou se forem discretos, a ordenação pode ser feita em $O(n)$. Essa estratégia é ótima uma vez que o tempo de espera total é dado pela fórmula:

$$T = n \cdot t_1 + (n-1) \cdot t_2 + \ldots + (n-k+1) \cdot t_k + \ldots + 2 \cdot t_{n-1} + 1 \cdot t_n$$

Onde t_1 é o tempo de serviço do primeiro cliente atendido, t_k é o tempo de serviço do k-ésimo cliente atendido e t_n é o tempo de serviço do último cliente atendido. Como os clientes que são atendidos primeiro possuem um multiplicador maior, para minimizar a soma basta colocar os clientes que são atendidos mais rapidamente no começo da fila.