Lista de Exercicios - Algoritmos Gulosos

1. Este é um problema que ocorre em análise automática de programas. Para um conjunto de variáveis x1, …, xn, são dadas algumas restrições de igualdade, da forma “xi = xj” e algumas restrições de desigualdade “xi ≠ xj”. Será que é possível satisfazer todas elas? Por exemplo, as restrições x1 = x2, x2 = x3, x3 = x4, x1 ≠ x4, não podem ser satisfeitas. Sendo assim, é possível utilizar o método guloso para elaborar um algoritmo eficiente que tome como entrada *m* restrições sobre *n* variáveis e decida se as restrições podem ser satisfeitas? Justifique.

R:

Não, pois neste problema não é admitido uma resposta parcial pois quaisquer restrições apresentadas podem influenciar todas as outras, sendo assim uma análise completa, pois mesmo que achemos uma uma solução viável pode haver uma restrição que a inviabilize.

1. Compressão de Huffman

Queremos armazenar em binário um código genético formado pelos símbolos A, C, T, G. Sabemos que a frequência em que cada um desses símbolos aparece é, respectivamente

fA = 0.55 fC = 0.05 fT = 0.15 fG = 0.25.

Qual a maneira mais compacta de representar o código genético?

• Codificação de tamanho fixo: Uma primeira idéia seria fazer A = 00, C = 01, T = 10 e G = 11.

• Codificação de tamanho variável: Outra ideia é usar uma quantidade variável de bits por símbolo

usando menos bits para símbolos que aparecem mais vezes na informação que queremos representar. Problema: Evitar ambiguidade (pense em um exemplo de codificação ambígua).

Solução: Nenhuma sequência de bits que representa um símbolo é prefixo de alguma outra sequência de bits que representa outro símbolo. Esse tipo de codificação é chamada livre de prefixos. Exemplo: A = 0, C = 100, T = 101 e G = 11.

1. Pergunta: Na codificação fixa, o que significa 00-00-11-10-01-11-00?

A A G T C G A

1. Pergunta: Na codificação variável, o que significa 0 0 11 101 100 11 0?

A A G T C G A

• Quantidade esperada de bits esperada para representar *n* símbolos na codificação fixa: 2n.

• Quantidade esperada de bits esperada para representar *n* símbolos na codificação variável: *n* (0.55 · 1 + 0.05 · 3 + 0.15 · 3 + 0.25 · 2) = 1.65*n*.

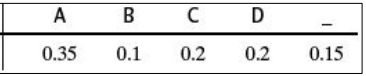
Como encontrar a codificação livre de prefixos ótima? Montando uma árvore binária a partir de escolhas gulosas, onde o custo é a frequência que o símbolo aparece.

Algoritmo:

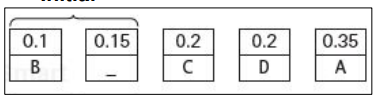
1. Comece criando *n* árvores com um único nó, rotulados com os símbolos do alfabeto a ser representado. Grave a frequência de cada símbolo para indicar o peso do nó.
2. Repita as operações a seguir até unir as sub-árvores em uma única árvore:
   1. Encontre as duas sub-árvores com os menores pesos (empates podem ser quebrados arbitrariamente)
   2. Faça-as serem as subárvores esquerda e direita de uma nova árvore, e grave a soma dos pesos na raiz da nova árvore como seu peso.
3. Ao final, atribua o valor 0 (zero) para as subárvores esquerdas e 1 para as subárvores direitas. O resultado do caminho da raiz até a folha dá o código correspondente de cada símbolo.

Exemplo:

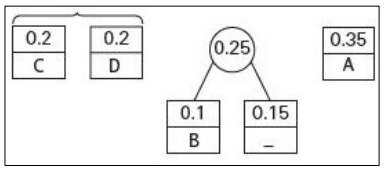
Imagine o alfabeto {A, B, C, D, -} com as frequências de acordo com a tabela abaixo:



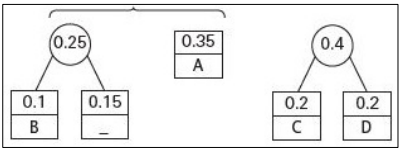
Inicialização:



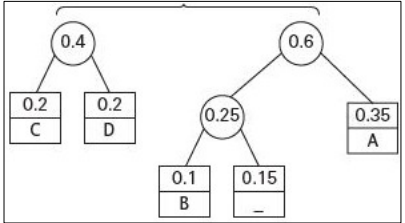
1ª Iteração:



2ª Iteração:



3ª Iteração:



4ª Iteração:

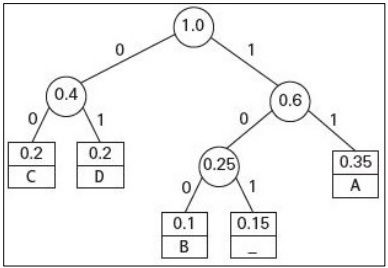
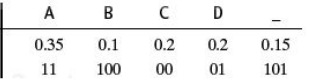


Tabela de símbolos resultante:



Esse algoritmo é conhecido como **compressão de Huffman** e é um dos algoritmos de compressão de arquivos mais conhecidos.

1. Execute (manualmente) o algoritmo para o alfabeto do DNA, mostrando o estado da floresta a cada passo e escrevendo a tabela resultante.

1 . Inicialização: Colocando as folhas ordem crescente de acordo com a frequência

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| C |  | T |  | G |  | A |
| 0,05 |  | 0,15 |  | 0,25 |  | 0,55 |

2. Como é uma árvore binaria pegaremos as folhas com menor frequência.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0,2 |  |  | 0,25 |  | 0,55 |
| 0,05 |  | 0,15 |  | G |  | A |
| C |  | T |  |  |  |  |

3-

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 0,45 |  |  |  | 0,55 |
|  |  |  |  |  |  |  | A |
|  | 0,2 |  |  |  | 0,25 |  |  |
| 0,05 |  | 0,15 |  |  | G |  |  |
| C |  | T |  |  |  |  |  |

4-

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 1 |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  | 0,45 |  |  |  | 0,55 |  |
|  |  |  |  |  |  |  | A |  |
|  | 0,2 |  |  |  | 0,25 |  |  |  |
| 0,05 |  | 0,15 |  |  | G |  |  |  |
| C |  | T |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

5- TABELA FINAL

|  |  |  |  |
| --- | --- | --- | --- |
| C | T | G | A |
| 000 | 001 | 01 | 1 |

1. Usando a tabela resultante, escreva as cadeias de bits resultantes para as sequências de DNA abaixo:
   1. AACTTGAATTACAAG

11000001001011100100110001101

* 1. TATAATCCGATTAAA

001100111001000000011001001111

* 1. ATTATTGTCAATGGA

10010011001001010010001100101011

1. Considerando que a tabela ASCII é uma tabela de codificação fixa, que usa 8 bits por símbolo, calcule, para cada sequência da questão anterior, qual foi a taxa de compressão obtida com a tabela de codificação resultante.

Bi = 120 – 100 x=2900/120 x≅24,17 espaços ocupado e ≅76,83% de compressão

29 – x

Bii=120 – 100 x=3000/120 x=25% espaços ocupado e 75% de compressão

30 – x

Bii=120 – 100 x=3200/120 x≅26,67 espaços ocupado e ≅73,33 de compressão

32 – x

1. **Questão bônus:** implemente um programa que faz a compressão de Huffman para o DNA, usando a tabela de codificação encontrada no exercício 3. O programa deverá receber um arquivo de texto como entrada, contendo *n* linhas, cada uma com uma sequência de DNA contendo *k* símbolos. O programa deverá imprimir uma linha para cada instância da entrada, contendo a sequência de bits resultante e a taxa de compressão atingida, com uma casa decimal, separados por um espaço. O fim da sequência de entrada é dado pelo fim do arquivo.

Exemplo de entrada:

AA

ATTG

CATTAGGAATT

T

Exemplo de saída (usando uma tabela aleatória):

00 87,5

01111101 83,3

1101011010110110001111 75,0

110 62,5

A = '1'

C = '000'

T = '001'

G = '01'

s = input()

sn = s.replace('A', A).replace('C', C).replace('T', T).replace('G', G)

print('{} {:.1f}'.format(sn, 100 - ((len(sn) \* 100) / (len(s) \* 8 ))))