**Anticompressor**

**João Vitor Boer Abitante**

Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

Porto Alegre – RS – Brasil

Joao.Abitante@edu.pucrs.br

***Resumo:*** *Este artigo apresenta a solução para calcular a quantidade de caracteres da string de um documento pós processo de anti-compressão, aonde no documento, há letras, e cada letra tem ou não, sua respectiva frase. Será apresentada a descrição do problema, sua modelagem, uma solução e pseudocódigos do algoritmo criado. Para concluir são testados 10 casos fornecidos e é feita a análise dos resultados obtidos. O respectivo artigo compõe a disciplina de Algoritmos e Estrutura de Dados II, da Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.*

# Introdução

Table

Description automatically generatedO problema a ser investigado pode ser resumido assim: temos dados comprimidos em um arquivo (Figura 1), que podem ser facilmente entendidos como um dicionário, onde o *char* é a chave e a *string* é o valor.

Figura 1. Caso informado no enunciado do trabalho.

Deve-se então, descomprimir e encontrar a a quantidade de letras da *string* final, ou seja, sempre que encontrarmos a letra *“u”* devemos transformá-la em *“mimimi”*. Para isso, segue-se três principais regras:

1. Encontrar o *char* e a *string* aonde inicia-se a substituição, afim de evitar loops.

2. Tratar corretamente as chaves que não contém valor, por exemplo, a letra *“m”*.

3. A descompressão é encerrada somente quando a *string* final não tiver nenhum *char* para ser substituído.

Por exemplo, seguindo estas regras obtemos como resultado:



O problema a ser resolvido é determinar a quantidade final de carateres da *string* e o tempo que levou para executar o programa. Segundo o enunciado do trabalho, este caso teria um total de 47 caracteres. Ao todo, são 10 arquivos de casos de teste fornecidos pelo professor que devemos descomprimir.

Afim de solucionar este problema, analisaremos uma solução possível que seja eficiente. Após, os resultados obtidos serão apresentados em uma tabela juntamente com as concluosões realizadas.

# Modelagem

Ao analisar o que foi solicitado podemos pensar em duas opções: fazer as substituições e no final calcular o tamanho da *string* resultante ou somar em uma variável o tamanho da *string* substituta sempre que for possível substituir. Tendo em vista essas duas possibilidades e a performance do código, escolhemos a segunda opção, pois na primeira seria necessário realizar um quantidade enorme de substituições que resultariam um custo de processamento imenso.

Considerando-se a escolha acima e a maneira como os casos estão escritos no arquivo, optamos por armazenar as informações em uma estrutura de lista (desconsiderando os *chars* em que não há respectiva *string)* aonde cada posição será um objeto de uma classe chamada Nodo, que contém os seguintes atributos: “sozinha”, “sequencia”, “primeiro” e “tamanho”. Estes correspondem respectivamente ao *char*, a *string*, um *boolean* que começará em True, e o tamanho da *string* atual.

class Node:

def construtor(sozinha, sequencia, primeiro, tamanho):

sozinha = sozinha

sequencia = sequencia

primeiro = primeiro

tamanho = tamanho

Logo, nossa lista ficaria assim:

|  |  |  |  |
| --- | --- | --- | --- |
| Nodo 1:  a  memimomu  True  8 | Nodo 2:  e  mimomu  False  6 | Nodo 3:  i  mooo  False  4 | Nodo 4:  u  mimimi  False  7 |

\*Lista

Os métodos implementados para encontrar a solução para cada um dos casos fornecidos serão descritos a seguir.

# Solução

O primeiro passo para resolver o desafio é realizar a leitura dos arquivos de casos de teste colocando os valores corretos na lista de classes. Para começar, precisamos ler as linhas e separar a letra sozinha de sua sequencia. Com está constatação, podemos propor uma resposta bastante simples: visto que cada linha é um vetor de *chars*, bastaria pegar a posição *linha[0]* para ser a letra inicial e consequentemente pegar o que sobrou (ignorando o espaço) para ser a frase correspondente (*linha[2:-1]).*

Porém, como foi citado na introdução, faz-se necessário desconsiderar os *chars* aonde não há respectiva *string.* E para isso, utiliza-se de uma condição que verifica a existencia da mesma. Caso a condição for verdadeira, o algoritmo adiciona um objeto na lista contendo a letra inicial encontrada, sua frase, um paramentro True (será explicado posteriormente), e o tamanho da frase correspondente. Um algoritmo implementando está idéia seria parecido com este:

abrir (‘casox’, ‘ler’) as arquivo:

para linha em arquivo:

if linha[2] != ‘\n’:

sozinha = linha[0]

sequencia = linha[2:-1]

tamanho = len(sequencia)

lista\_classes.adicionar(Nodo(sozinha, sequencia, True, tamanho))

O próximo passo é encontrar qual é a letra incial do processo de descompressão, e para isso seguimos a ideia de que a letra não pode estar contida em nenhuma das sequencias do arquivo. Sabendo disso, podemos pensar em uma alternativa: realizar a verificação de cada letra com todas as sequencias obtidas. Se essa verificação for válida, ou seja, a letra está em alguma das sequências, alteramos o parâmetro *primeiro* para False, caso contrário, permanecerá em True. Está solução seria assim:

para i < tamanho(lista):

para j linha tamanho(lista):

if lista[i].sozinha in lista[j].sequencia:

lista[i].primeiro = False

# Função Caso

Essa função pode ser separada em três partes: leitura, demarcação do início e busca do resultado esperado.

No momento da leitura foi implementado um *try except*, para prevenir erros na leitura dos arquivos, inibindo a execução deles, sem prejudicar a anticompressão dos restantes.

try:...

except: #Caso ocorra algum erro na leitura do arquivo print(' Erro na leitura do arquivo:',caso,".")

**Testes:**

O teste dessa implementação se baseou em dois arquivos, um sendo inexistente e o outro sem a presença 5de espaço entre as letras indificadoras e as palavras substitutas, figuras 2 e 3. Retornando em ambos os casos a mensagem de erro esperada, figura 4.

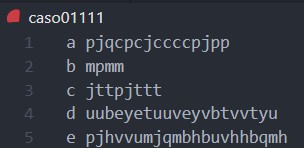


Figura 2. “Caso01”, com seu nome alterado.

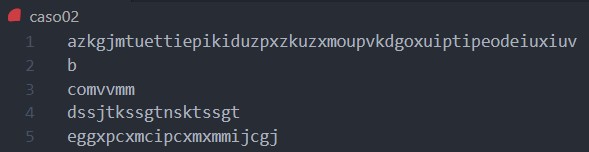


Figura 3. “Caso 02”, sem os espaçamentos entre as partições da linha.



Figura 4. Mensagem de erro gerada pelo teste.

Após isso é registrado as linhas do documento, na classe Linha, somente as que tiverem tanto a letra quanto a frase de substituição, facilitando o trabalho que será exercido pela função Soma.

#Guarda as letras e as palavras em um vetor

with open(caso,'r') as arquivo: #abre o respectivo arquivo vetor = [] #vetor que será o vetor para a classe linhas for line in arquivo: #para cada linha do arquivo

if (line[2]) != '\n': #quando a linha tiver a palavra de substituição

vetor.append((Linhas(line[0],line[2:], True,0))) #registra a linha no vetor

Após o registro de todas as linhas necessárias, é feita a verificação da linha que deverá ser a primeira a ser lida, devido ao fato anteriormente comentado. A identificação dessa linha é realizada por meio da confirmação de chamamentos das letras, em todas as palavras registradas, sendo assim quando uma letra não for encontrada, após essa verificação, é categorizada como a inicial.

#Demarca qual é o inicial

for i in range(len(vetor)):#vetor das palavras for x in range(len(vetor)):#vetor das letras

for j in range(len(vetor[i].palavra)):#vetor dos chars da palavra if vetor[x].letra==(vetor[i].palavra[j]):#caso essa letra for encontrada vetor[x].primeiro=False#é registrada como não sendo a primária

Caso identificado a “linha primária”, o processo de anticompressão da frase de substituição dela é iniciado, nesse momento é chamada a função Soma, que possibilitará a conclusão do problema. Porém se não for constatada a existência de uma linha primária, é informado ao usuário a inexistência da linha primária e não ocorre a soma do respectivo caso. Essa implementação é para prevenir que arquivos irregulares, possam gerar erros ou serem retornados com valores errôneos, fato ocorrido em testes realizados durante a criação do programa.

#Encontra o primeiro char da linha inicial soma = 0 #valor da soma dos chars finais

happen = False #Para casos que não tenham palavras iniciais (todas as frases se chamam) for i in range(len(vetor)): #passa por todas as letras

if vetor[i].primeiro==True: #encontra a respectiva primeira linha que tem que ser lida

for j in range(len(vetor[i].palavra)): #passa por todos os chars da palavra soma+=Soma(vetor,vetor[i].palavra[j]) #salvando o valor de cada char na soma total

happen = True

if happen == False: #Não tem palavra inicial (todas as frases se chamam)

print(' Arquivo',caso,'sem uma palavra inicial, possivel erro no arquivo.') # Arquivos sem a primeira palavra podem por gerar erros ou resultados finais falhos else: print(" ",caso," tem o valor de:",soma) #resposta final

**Teste:**

O teste dessa implementação se baseou na alteração de um arquivo, onde teve a adição da letra da sua “linha primária”, a uma frase antes existente, como mostrado na figura 5, gerando a saída da figura 6.

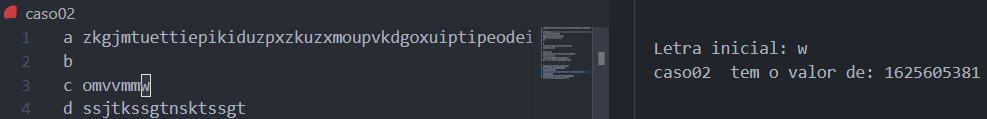


Figura 5. “Caso02”, com a letra inicial presente em uma outra linha.



Figura 6. Mensagem de erro gerada no teste antes apresentado.

O retorno de cada teste entregará o mesmo padrão de resposta, somente diferenciando as informações do caso realizado e seu valor final obtido. Por exemplo o resultado obtido no “caso01”, como pode ser observado a seguir.

caso01 tem o valor de: 17168630

# Código da Função Caso

Como a função foi repartida em pedaços específicos, para facilitar a explicação de sua funcionalidade, como um todo, disponibilizarei a função inteira abaixo, possibilitando a melhor verificação de como foi formatada e criada.

def Caso(caso): try:

#Guarda as letras e as palavras em um vetor

with open(caso,'r')as arquivo: #abre o respectivo arquivo vetor = [] #vetor que será o vetor para a classe linhas for line in arquivo: #para cada linha do arquivo

if (line[2]) != '\n': #quando a linha tiver a palavra de substituição vetor.append((Linhas(line[0],line[2:], True,0))) #registra a linha no vetor

#Demarca qual é o inicial

for i in range(len(vetor)): #vetor das palavras for x in range(len(vetor)): #vetor das letras

for j in range(len(vetor[i].palavra)): #vetor dos chars da palavra if vetor[x].letra==(vetor[i].palavra[j]):#caso essa letra for encontrada vetor[x].primeiro=False #é registrada como não sendo a primária

#Encontra o primeiro char da linha inicial soma = 0 #valor da soma dos chars finais

happen = False #Para casos que não tenham palavras iniciais (todas as frases se chamam)

for i in range(len(vetor)): #passa por todas as letras

if vetor[i].primeiro==True: #encontra a respectiva primeira linha que tem que ser lida

for j in range(len(vetor[i].palavra)): #passa por todos os chars da palavra soma+=Soma(vetor,vetor[i].palavra[j]) #salvando o valor de cada char na soma total

happen = True

if happen == False: #Não tem palavra inicial (todas as frases se chamam)

print(' Arquivo',caso,'sem uma palavra inicial, possivel erro no arquivo.') # Arquivos sem a primeira palavra podem por gerar erros ou resultados finais falhos else: print(" ",caso," tem o valor de:",soma) #resposta final except: #Caso ocorra algum erro na leitura do arquivo print(' Erro na leitura do arquivo:',caso,".")

# Função Soma

A funcionalidade de Soma se baseia em uma série de recursões, que descobrem e registram a quantidade de caracteres que cada sequência das letras gera.

Os valores das frases são decorrentes da verificação do registro dos caracteres presentes nas frases de substituição, de cada objeto de Linhas e quando alguma letra não for encontrada dentro da classe é somado “1”, ao valor da respectiva sequência. Ao término da decodificação da frase o seu total é salvo, para facilitar o processo nos próximos chamamentos, retornando imediatamente seu valor, caso isso ocorra.

def Soma(vetor,letrinha): soma = 0 #valor da soma

achou = False #caso seja um char registrado na classe Linhas for i in range(len(vetor)): #verifica todas as letras registradas if(vetor[i].letra==letrinha): #quando encontra a letra

if vetor[i].valor == 0: #verifica se o valor dela ainda não foi descoberta achou=True #informa que o esse char foi encontrado

for j in range(len(vetor[i].palavra)): # verifica o tamanho da palavra soma+=Soma(vetor,vetor[i].palavra[j]) #Descobre o valor total do char vetor[i].valor=soma #adiciona o valor no char i=len(vetor)#encerra a verificação das letras else:#caso o valor do char já tenha sido registrado achou=True #informa que o esse char foi encontrado soma=vetor[i].valor #adiciona seu valor na soma

if (achou==False and letrinha!= '\n'): #caso a letra não esteja registrada, será adicionada a soma soma=1

return soma

# Resultados

Para melhor análise dos dados presentes, dentro de cada caso de teste entregue, criei as Tabelas 1 e 2. Nelas estão presentes: nome, quantidade de palavras substitutas, caracteres das frases e

caracteres finais, presentes e gerados por cada um dos dez testes realizados.

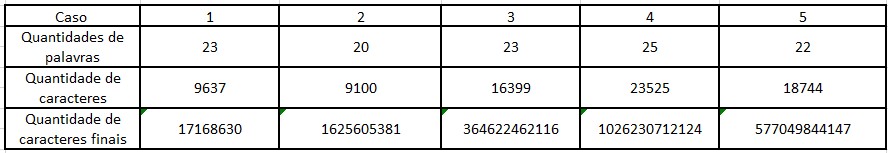


Tabela 1. Quantidades de caracteres finas encontrados.

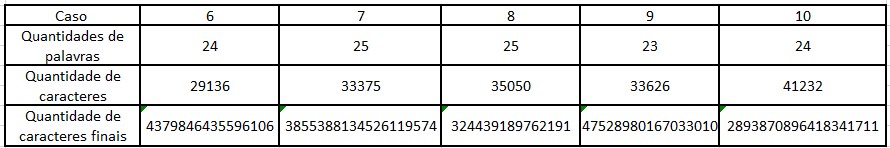


Tabela 2. Continuação dos dados dos casos entregues.

Com base nos dados antes apresentados, torna-se possível perceber que o número de caracteres tem um aumento influenciado, principalmente pela quantidade de palavras de substituição e posteriormente, pela quantidade de caracteres das palavras. Devido a esse motivo torna-se possível explicar o porquê do resultado do “caso7” ser maior do que o “caso10”, sendo que em comparação, o último caso tem uma quantidade maior de caracteres que o outro.

# Conclusão

Esse projeto demonstra que muitas vezes é mais sensato utilizar um código eficiente e simples ao invés de um complexo e com um menor rendimento. Nas primeiras versões, tentei implementar ideias que envolviam manipular os dados de uma *string* e só no final adquirir o resultado esperado, mas após diversos testes, percebi que estava complicando meu programa sem motivo. Devido a isso, simplifiquei o programa, trocando a *string* por um contador que soma os resultados obtidos de cada manipulação que seria realizada, obtendo assim um programa que chega ao mesmo resultado sem gerar erros de excesso de memória e em um menor espaço de tempo. Conseguindo até mesmo executar os dez casos entregues na proposta, em menos de um segundo, como é possível ver no tempo de execução da figura 7.

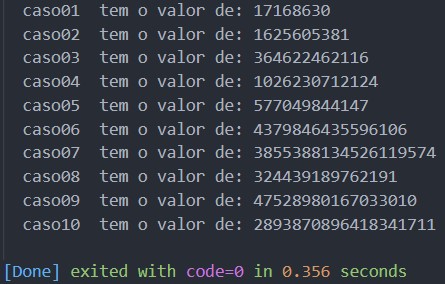


Figura 7. Execução do anticompressor.