Anticompressor

Rafael Schaker Kopczynski da Rosa

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

Av. Ipiranga, 6681 – 90.619-900 – Porto Alegre – RS – Brasil

Rafael.Schaker01@edu.pucrs.br

Resumo: *Este artigo apresenta a solução para calcular a quantidade de caracteres compressados em um documento, contendo o uma letra do alfabeto que representa uma frase. Será apresentada a descrição do problema, juntamente com a sua análise, raciocínio da resolução e o pseudocódigo do algoritmo implementado, usando manipulação de um vetor de classes. Para concluir será apresentada a resolução de dez "casos" fornecidos. O respectivo artigo compõe a disciplina de Algoritmos e Estrutura de Dados II, da PUCRS.*

Introdução

Foram entregues no total, dez casos de testes, todos sendo documentos contendo o alfabeto e tendo em algumas de suas letras a presença de *strings* ao lado, como é representado na Figura 1.

```
a pjqcpcjccccpjpp
b mpmm
c jttpjttt
d uubeyetuuveyvbtvvtyu
e pjhvvumjqmbhbuvhhbqmh
f pvqtatkvqwvrrroywoaggtgudlwgkgu
g umhphvtmbhutjpctuhuhjc
h ppkcqkcqck
i qppppqq
j pmqmpqqqmbmp
k bmmbm
l nvjrjiiiimjizgrzmvyivvxivnynnxr
m ttt
n evpvettjiyweevpypjpupjhitpwttwpvd
o pgeccktyemekmye
p
q m
r woegwtgjanntjvwjneedjvndzavnod
s
t
u bbccqmmbqqcq
v bmpmm
w bkqyitpppuupyeuyuduiydqtediptbedmytzjmepjy
x zavjvdiiiagyvjjwwyyyvcvgagwazci
y btpbputthuqttpthth
z jpmujyymmoypkjkiyoym
```

Figura 1. Representação do "caso01".

O suposto problema consiste em criar um anticompressor (decodificador) de texto. O texto a ser decodificado se encontra em um arquivo parecido com a representação acima, onde cada linha tem uma letra do alfabeto e sua respectiva palavra ou frase de equivalência.

A decodificação ocorre na substituição das letras pelas suas frases correspondentes e somente é encerrada, quando a palavra final não tiver nenhum caractere que possa ser descomprimido.

Cada caso pode conter ou não *strings* de quantidades indeterminadas, podendo assim gerar resoluções pequenas ou até gigantescas.

Quando uma letra não tem uma *string* na sua mesma linha é considerada uma letra final, sendo assim ela sobrará na sequência final gerada, após as inúmeras substituições que deverão ser feitas para a resolução de seu respectivo "caso".

Análise e modelagem do problema

Após a realização de alguns testes prévios e pesquisas nos arquivos que foram entregues juntamente com a proposta, foi possível notar, que seria necessário "forçar" a leitura de uma respectiva linha em primeiro lugar, para assim ocorrer a devida decodificação. Essa obrigatoriedade é devido ao fato de que certas letras com palavras substitutas não estarem presentes nas frases de nenhuma das linhas do arquivo, não sendo decodificadas e prejudicando o resultado esperado.

Para obter a quantidade de caracteres finais da palavra decodificada dos arquivos, categorizados como "casos", decidi criar um anticompressor, com a linguagem "*python*", uma classe nomeada de "Linhas" e um vetor com seus objetos. A resolução desse problema foi construída no ambiente de programação *VSCode*.

Solução

Ao realizar a leitura do arquivo gravei os dados de cada linha, na classe Linhas separando-os entre "letra identificadora" e "palavra de substituição".

Com testes antes realizados, criei mais duas variáveis para a classe Linhas, sendo elas: "primeiro" e "valor". "Primeiro" informará se alguma das linhas deve ser descompactada antes das outras, fato anteriormente registrado e "valor" terá o intuito de aumentar a eficiência do meu código, guardando a quantidade de caracteres finais (que não podem mais ser descomprimidos), de cada "letra identificadora". Assim requisitando descompressão das palavras somente uma única vez e nas próximas vezes em que for requisitado, retornará à quantidade presente em valor.

```
class Linhas:#cada letra que será substituida por uma palavra é salva aqui
      def __init__(self,letra,palavra,primeiro,valor):
        self.letra = letra
        self.palavra = palavra
        self.primeiro = primeiro
        self.valor  = valor
```

Para a conclusão do anticompressor, criei algumas funções, além da “Main”, sendo elas a “Soma” e “Caso”.

A quantidade de caracteres finais de cada caso, entregues na proposta do programa, serão apresentadas e comentadas no final do artigo.

Função Main

Essa função foi utilizada para realizar cada caso entregue na realização desta tarefa, implementado por um *for* de zero a dez e um *switch case*, que continha um chamamento para a realização de cada caso.

```
caso ='';
for i in range(10):
    match i:
        case 0:
            caso='caso01'
        case 1:
            caso='caso02'
        case 2:
            caso='caso03'
        case 3:
            caso='caso04'
        case 4:
            caso='caso05'
        case 5:
            caso='caso06'
        case 6:
            caso='caso07'
        case 7:
            caso='caso08'
        case 8:
            caso='caso09'
        case 9:
            caso='caso10'
    Caso(caso)
```

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 de 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.

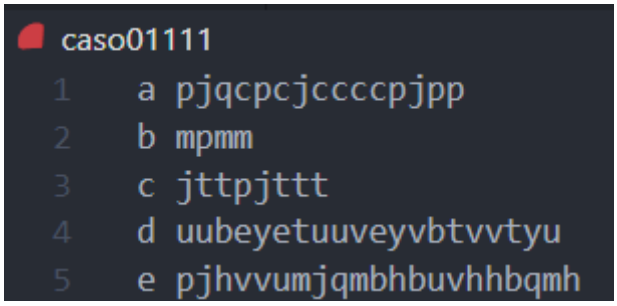

Figura 2. “Caso01”, com seu nome alterado.

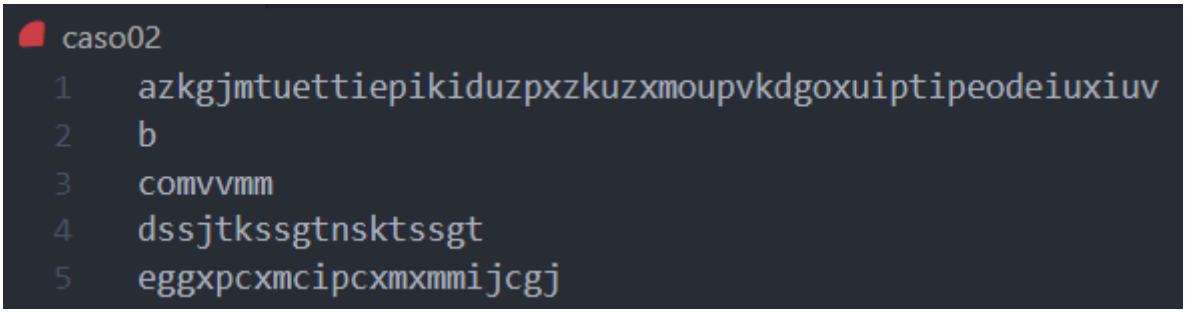

Figura 3. “Caso 02”, sem os espaçamentos entre as partições da linha.

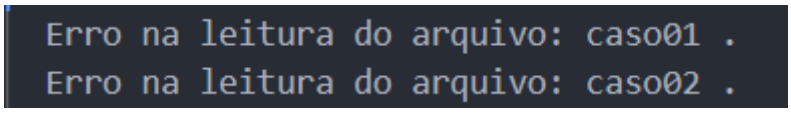

Figura 4. Mensagem de erro gerada pelo teste.

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.

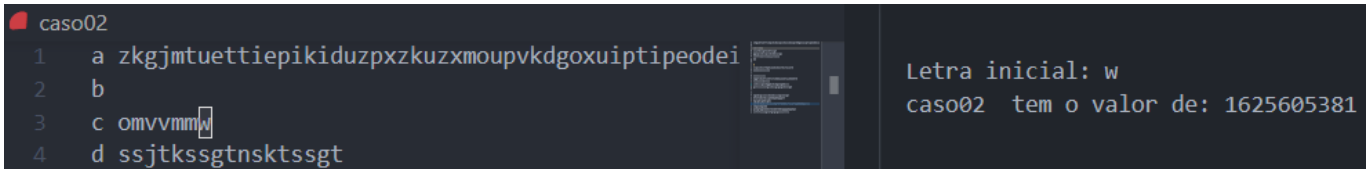

Figura 5. “Caso02”, com a letra inicial presente em uma outra linha.

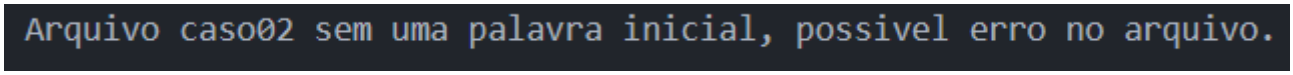

Figura 6. Mensagem de erro gerada no teste antes apresentado.

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.

Caso	1	2	3	4	5
Quantidades de palavras	23	20	23	25	22
Quantidade de caracteres	9637	9100	16399	23525	18744
Quantidade de caracteres finais	17168630	1625605381	364622462116	1026230712124	577049844147

Tabela 1. Quantidades de caracteres finas encontrados.

Caso	6	7	8	9	10
Quantidades de palavras	24	25	25	23	24
Quantidade de caracteres	29136	33375	35050	33626	41232
Quantidade de caracteres finais	4379846435596106	3855388134526119574	324439189762191	47528980167033010	2893870896418341711

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.

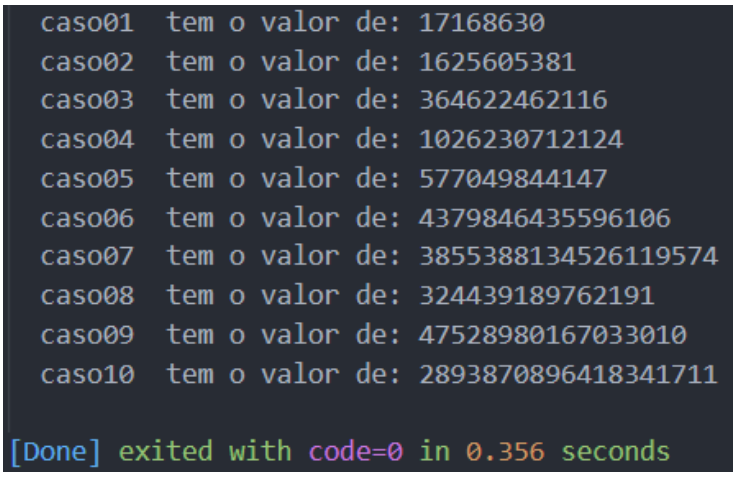

Figura 7. Execução do anticompressor.