Relatório do Trabalho

**Alunos:** Luis Henrique Santos Nascimento; João Victor Figueiredo Fernandes Santana; Rivaldo Leite Dos Santos Neto.

1. **Parte Um:**

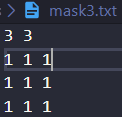
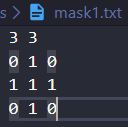
Para a primeira parte do trabalho da matéria tivemos como tarefa fazer um programa que recebe como entrada uma imagem com um texto no formato PBM e tem a função de remover o ruído dessa imagem para criar outro arquivo PBM que possui o conteúdo da imagem de entrada sem o ruído.

Inicialmente pensamos em remover o ruído usando abertura para remover os ruídos externos e fechamento para tirar os pontinhos brancos das letras. Como os ruídos externos eram formados por apenas 1 pixel ou 2 formando uma linha na vertical ou horizontal foram utilizadas as mascaras: mask2.txt e a mask4.txt que são mascaras 3x3 com 2 pixels na horizontal e vertical respectivamente. Então fizemos 2 aberturas uma com cada uma dessas 2 mascaras, posteriormente usamos 2 fechamentos cada um com uma das mesmas mascaras usadas na abertura pra remover os pontos brancos nas letras, fechar os espaços em branco dentro das letras.

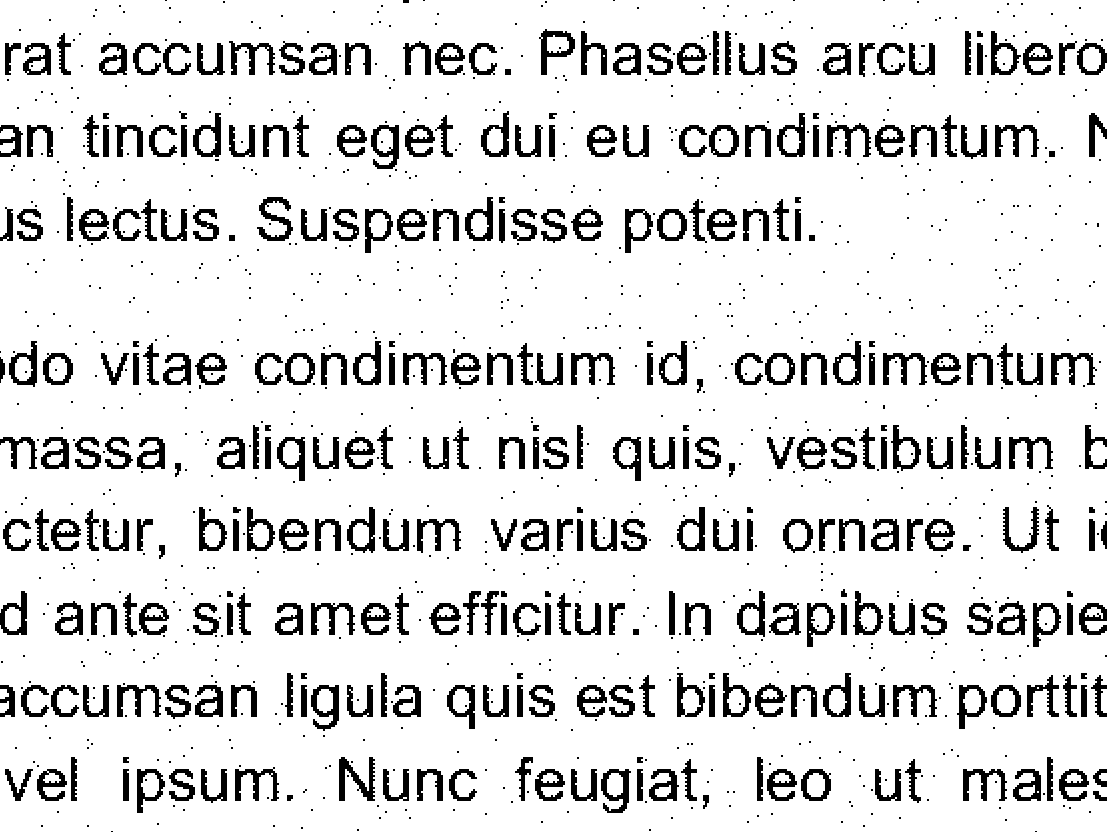
Como depois dessas operações algumas letras ficaram com uma fratura resolvemos inverter as operações, primeiro os fechamentos para fechar os espaços em branco das letras e depois a abertura cara remover o ruído externo. Essa inversão nas operações sanou o problema numa imagem que estávamos testando, porém ao usar essa mesma operação em outra imagem algumas inconsistências permaneceram em algumas letras.

Então resolvemos mudar de abordagem, usamos a mediana com a mascara mask3.txt de tamanho 3x3 com 9 pixels para remover o ruído. Como algumas bordas das letras ficaram mal formadas, com alguns espaços em branco entre os pixels, decidimos fazer uma dilatação com a mascara mask1.txt de tamanho 3x3 com formato de cruz para melhorar o formato das letras consertando as bordas, em seguida usamos 2 aberturas uma com a mask4.txt e outra com a mask2.txt para romper com as pequenas pontes entre as letras, tanto as horizontais quanto as verticais, porém algumas pontes restaram, mas essas são pontes que em média possuem 4 pixels e se usássemos uma nova abertura com uma mascara maior para rompe-las notamos que algumas letras seriam muito prejudicadas ficando com grandes fraturas, então decidimos manter essas pontes.

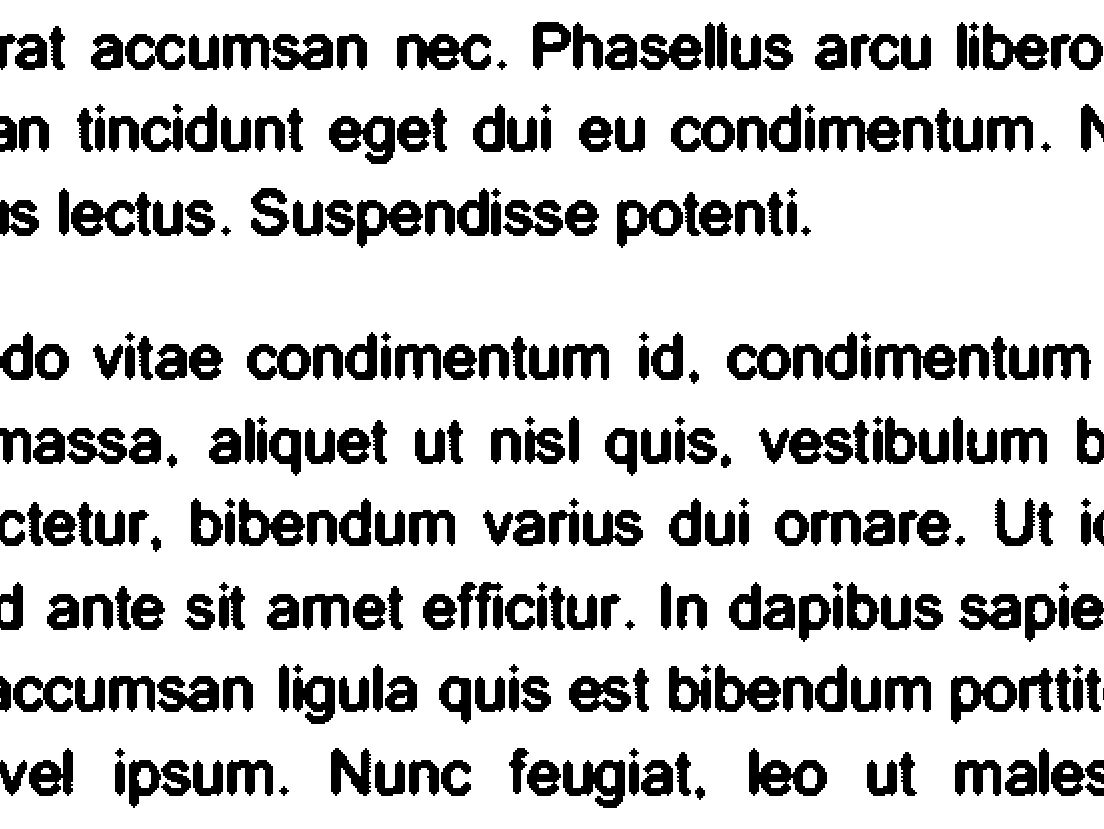
No processo várias outras tentativas foram feitas para ver qual seria a melhor combinação de operações e mascaras para tirar os ruídos e deixar as letras bem formadas e sem pontes, ao final percebemos que apenas precisávamos tirar o ruído sem se importar com o formato das letras ou se existia pontes entre elas então fizemos uma última alteração no código mantendo as operações de mediana e dilatação e removendo as duas operações de abertura.



**mask1.txt mask3.txt**

****

**Imagem com ruído**

****

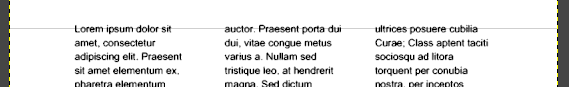
**Imagem de saída do programa.**

1. **Parte Dois:**

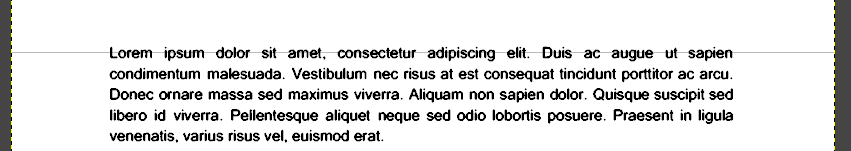
Para a parte dois do projeto tivemos como objetivo acrescentar no programa funções que contassem quantas colunas, linhas e palavras existiam na imagem, além de circular as palavras.

Começamos fazendo a função para contar as colunas, a ideia inicial foi posicionar uma lista no centro da primeira linha da imagem, sendo que essa lista possui tamanho menor do que o espaço que há entre uma coluna e outra, assim a lista inicia com 0 em todas as posições e conforme ela se movimenta horizontalmente e chega nas palavras onde existem pixels pretos, de valor 1, ela coloca dentro dela esses pixels sobre os quais “passa por cima” durante sua trajetória.

Durante sua trajetória, seu peso (a soma de todos os valores nela presentes) é verificado, caso o peso era 0 e foi alterado significa que o leitor (a lista) chegou a uma palavra, como ele é grande seu peso nunca volta a ser zero dentro de uma coluna. Porém ele é menor do que o espaço entre as colunas, assim, se o leitor tinha peso maior que 0 e agora é 0 significa que ela acabou de sair de uma coluna, logo o valor de colunas é incrementado. Nos também aproveitamos a parte da função em que o leitor detecta uma coluna, seu peso era 0 e agora é maior que 0, para guardar o ponto onde as colunas começam para utilizar nas funções de contar linhas e palavras.



**Trajetória do leitor usado para contar colunas.**

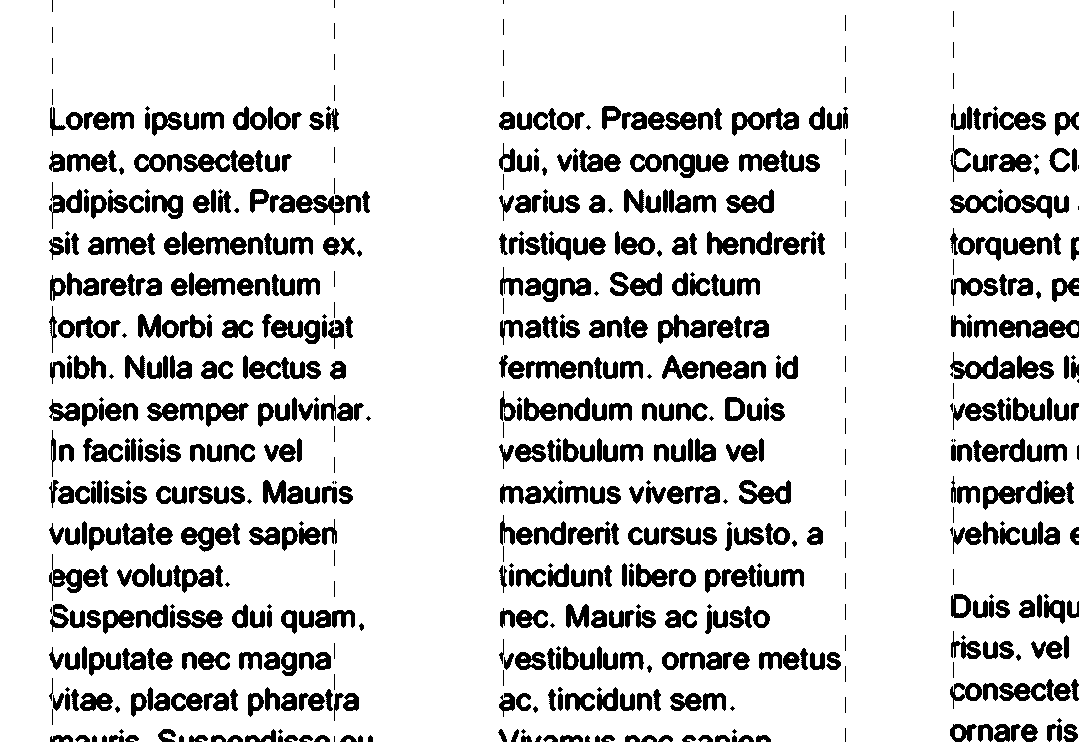


**Trajetória do leitor usado para contar colunas.**

Usando a mesma ideia usada para contar colunas, usando peso atual e peso anterior do leitor, fizemos a função de contar linhas. Nesse caso usamos um leitor menor, visto que o espaço entre as linhas são bem menores do que o espaço entre as colunas e esse se move na vertical.

Usamos os pontos conseguidos na função para contar colunas no momento de posicionar o leitor. O leitor era posicionado acima da primeira linha de cada coluna, uma coluna por vez, ao descer até o final do texto ele era reposicionado no inicio da próxima coluna para repetir a leitura nessa outra coluna. O objetivo era achar a coluna que possuía mais linhas. Um problema encontrado nessa função foi por conta dos parágrafos, que pulavam uma linha mais outra coluna poderia ter uma linha no lugar do parágrafo, assim o programa não considerava essa linha no número total de linhas da imagem.

Para resolver isso usamos uma *string* para verificar o quanto o leitor desceu sem ler nada, no caso dele detectar uma linha vazia do parágrafo, assim ao detectar a linha vazia o leitor pula horizontalmente para o ponto onde está a próxima coluna para averiguar se nessa outra coluna há uma linha exatamente onde existe um espaço em branco na coluna anterior, se não houver linha ele pula horizontalmente para próxima coluna até que todas as colunas sejam verificadas naquele ponto. Depois da verificação ele retorna a posição onde ele estava (espaço em branco) e continua o processo, sendo que se na verificação alguma das colunas vizinha tinha uma linha naquele ponto ele agora considera aquela linha na conta de linhas totais.

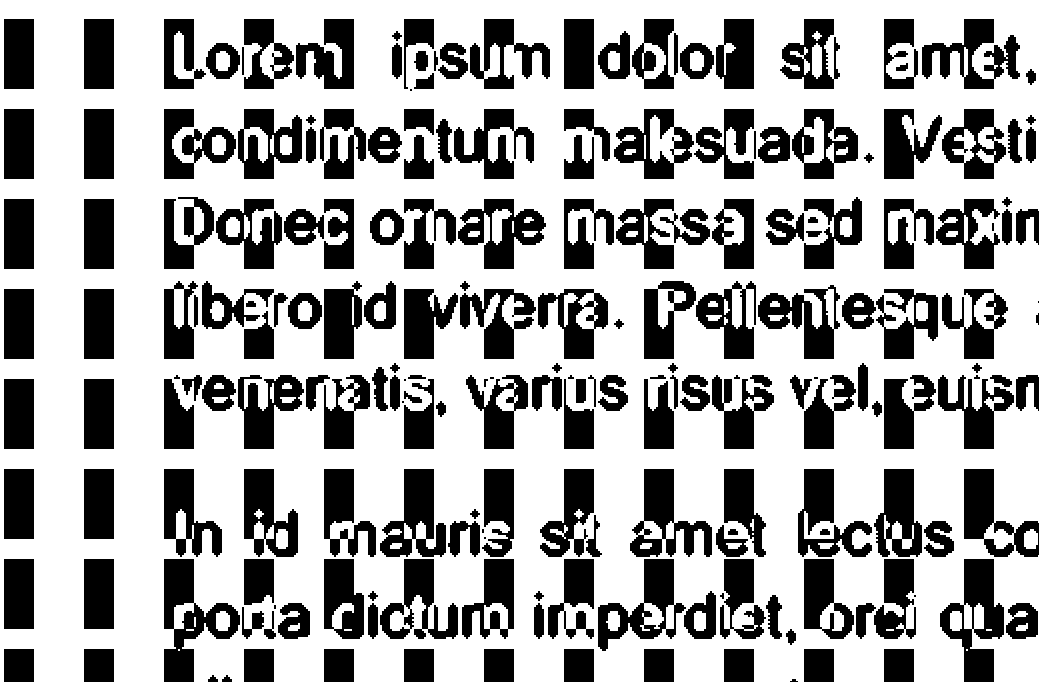


**Imagem do teste para conferir a trajetória do leitor utilizado para contar linhas.**

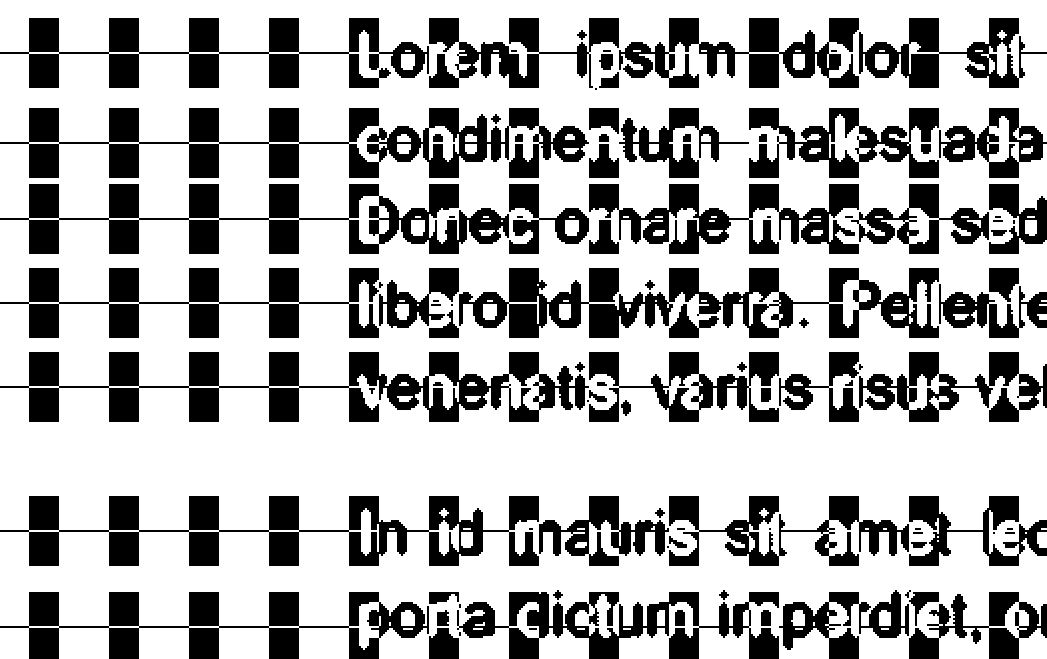
Continuando com a linha de ideia das funções anteriores montamos a função para contar palavras, porém dessa vez não utilizamos uma lista como leitor, mas sim uma matriz, pois precisaríamos da altura e da largura da palavra.

Da mesma forma que a função que conta as colunas guardava a posição das colunas, nos usamos a função de contar linhas para armazenar numa lista os pontos onde as linhas estavam para assim ter a posição central de cada linha de cada coluna e com isso posicionar o leitor exatamente em cima das palavras para percorrer cada uma das linhas das colunas.

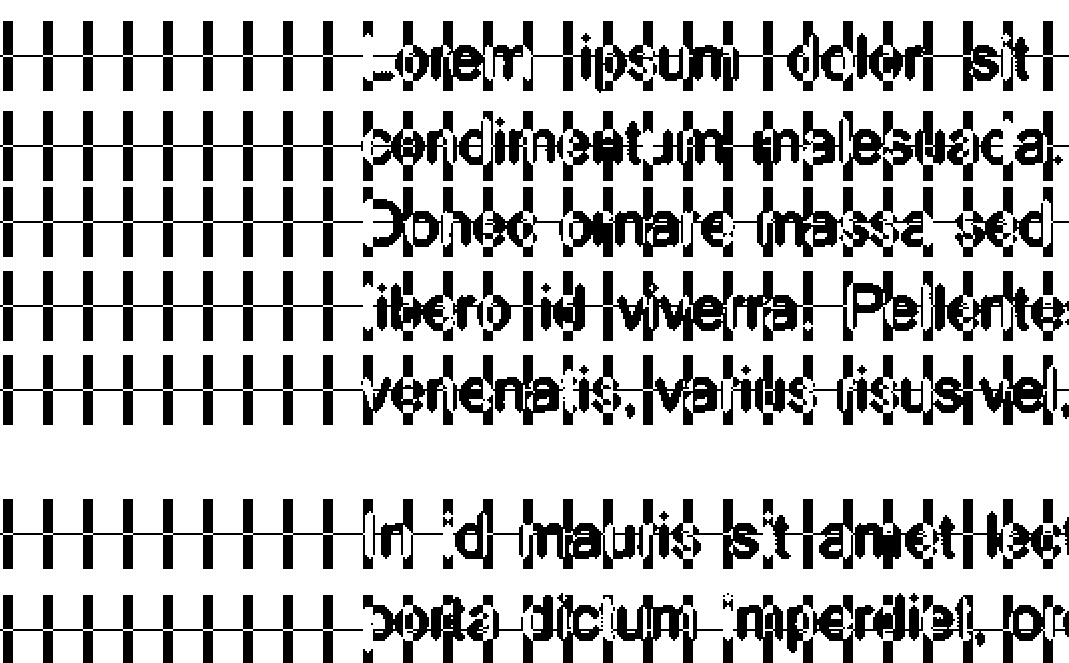
Assim com o leitor corretamente posicionado ele percorre todas as linhas da primeira coluna e vai para a próxima coluna e repete o procedimento até verificar todas as linhas. O processo de contar as palavras, assim como as funções anteriores se baseia no peso atual e anterior do leitor. Se era 0 e agora é maior que 0 o leitor chegou numa palavra. Se antes era maior que 0 e agora é 0 o leitor acabou de sair da palavra, então o número de palavras é incrementado em 1. Com isso nós também conseguimos os pontos x e y iniciais e finais das palavras que foram armazenados para serem usados pela função que circula as palavras.



**Tentativa de posicionar o leitor para pegar as palavras de forma completa.**

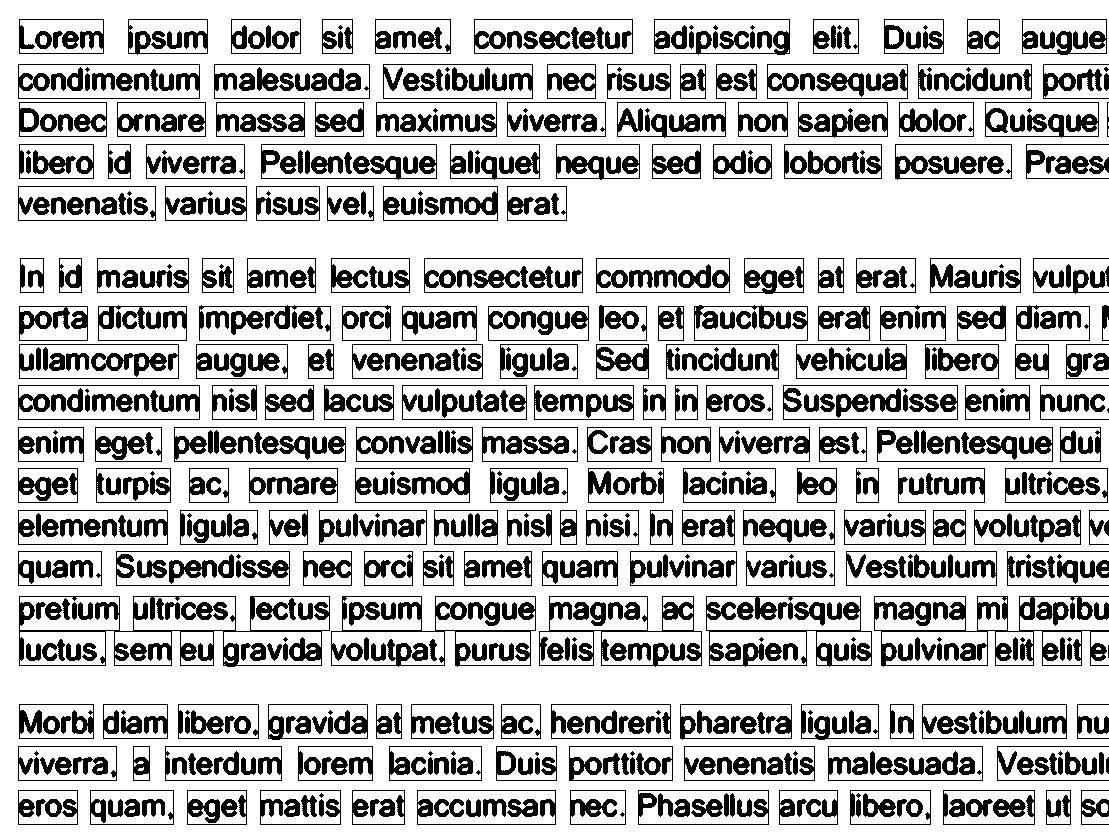


**Utilizando os pontos centrais das linhas para posicionar o leitor.**

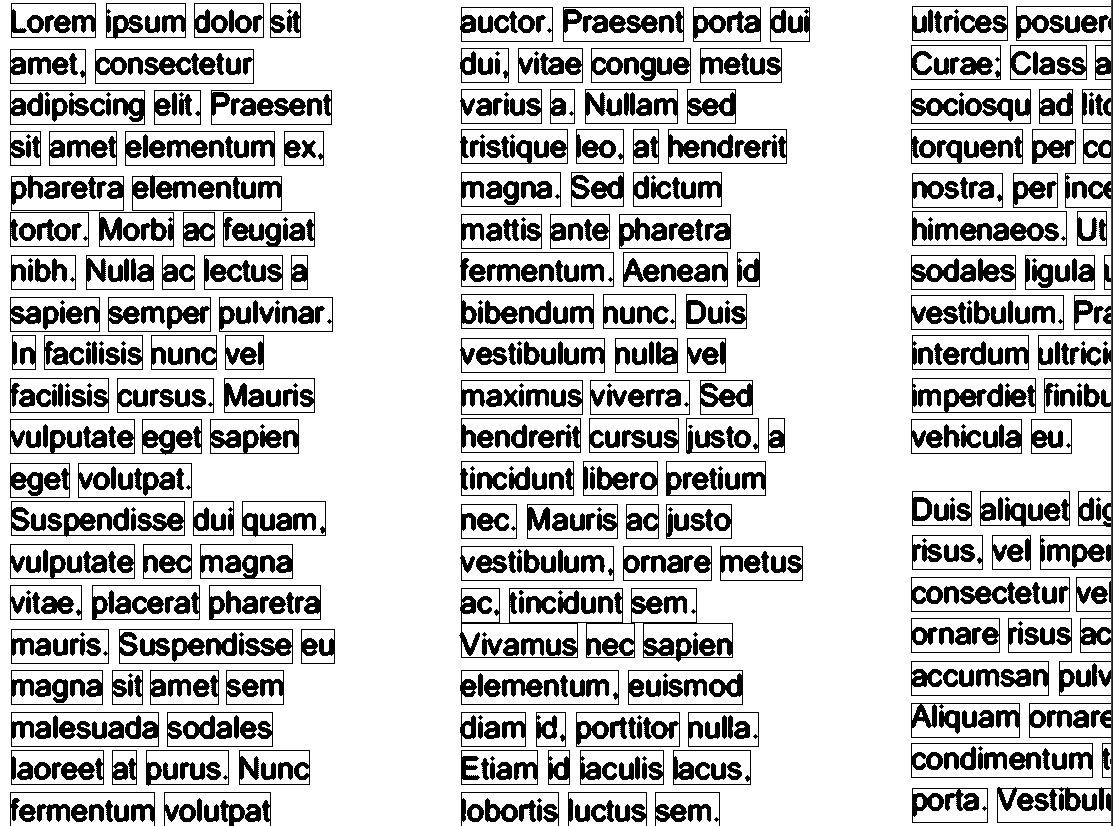


**Leitor ajustado para caber nos espaços entre as palavras e fazer a contagem.**

Por fim fizemos a função que circula as palavras usando o conjunto de pontos x e y conseguidos pela função que conta as palavras.



**Imagem 01 Final.**



**Imagem 02 Final.**



**Imagem 03 Final.**