ÍNDICE REMISSIVO

Feito por: Zeus Moreira de Lima Pereira.

Data: 21/11/2020.

1. Resolução

A ideia de resolução para o programa "Índice remissivo" é baseada em lermos determinados textos com no máximo 10000 caracteres cada, então o problema é dividido em dois.

1º) Verificar quantas palavras, linhas e parágrafos há no texto.

2º) Ler uma palavra e dizer o seu índice remissivo (Paragrafo, linha e coluna).

Isso é repetido para "n" casos de teste.

2. Funcionamento do algoritmo

Ao criarmos a estrutura de dados para a proposta do problema que nos fornece dados acerca das células da lista primária e secundária devemos criar um vetor de listas preenchendo essas listas com células que possuem dados de entrada e de processamento para solucionar o enunciado.

2.1) Criar as estruturas de dados: Estruturas *"first"* e *"second"*, sendo a lista primária e secundária respectivamente. Para o problema temos um vetor de ponteiros *"tab"* e para cada índice esse vetor *"tab"* aponta para uma lista de estrutura "*first"* que possui uma chave palavra dita como *"termo"*, uma chave com número de ocorrências dessa palavra dita como *"casos"*, uma chave que aponta para a lista secundária chamada de *"inicio"* e uma chave que aponta para a última célula dessa lista secundária chamada de *"fim"*. A lista secundária é composta por informações sobre a chave palavra da lista primária, sendo suas chaves: *"pg, cl e ln",* que representam o parágrafo, a coluna e a linha onde a palavra está dentro do texto fornecido.

2.2) Tratamento da entrada de dados: Após a leitura do número de casos de teste e de uma quebra de linha a preocupação agora é ler um texto de no máximo 10000 char's, para esse processo é reservado na memória um espaço do tamanho MAX para uma string. Após isso é efetuado a leitura da string dita como *"texto"* e desse modo é utilizado o tamanho do *"texto"* para a realocação desse espaço na memória, tendo como objetivo economizar. Depois disso é efetuada a leitura de outra quebra de linha, logo após há a leitura do número "n" de palavras do índice remissivo e consequentemente é fornecida as "n" palavras dita como *"str"*.

2.3) Processamento dos dados e o problema: Com o número "n" recebido é necessário descobrir o menor primo maior que "n" e esse primo será a quantidade de elemento do vetor de ponteiros para lista *"tab",* após isso é criada a lista "tab" e todas as cabeças com a função ***"tabv()*** ". A alocação e o recebimento da *"str" já* foram feitos e com isso declaramos e alocamos as células da lista primária e secundária, *"nova"* e "*begin"* respectivamente. Então preenchemos a célula *"nova"* usando a função ***"preencherF()"*** e para adicionarmos *"nova"* em *"tab"* necessitamos saber em que índice de *"tab"* essa célula ficará apropriada pra isso é utilizada a função ***"apropriado()"*** que devolve o índice necessário, logo depois utilizamos a função ***"adiciona()"*** para adicionar *"nova"* a *"tab".* Para o processamento da célula *"begin"* que tem cabeça *"head"* é necessário saber o local em que *"str"* está e para isso é usado a função ***"strstr()"***, da biblioteca <string.h>, após encontrada a palavra é chamada a função ***"preencherS()"*** com as funções ***"pag()"***, ***"col()"*** e ***"lin()"*** contidas para preencher *"begin"* e ***"adiciona2()"*** para adicionar *"begin"* em *"nova->inicio".* Todo o procedimento da célula *"begin"* está dentro de um loop que repete o número de vezes em que essa palavra está no *"texto"* para adicionar mais uma célula *"begin"* com as informações da segunda ocorrência da palavra se tiver.

2.4) Impressão e limpeza de memória: A impressão baseia em imprimir o número de palavras, número de linhas e parágrafos do *"texto"* e logo depois o número de *"str"* sendo a impressão todas as *"str"* digitadas, o número de repetições dessas *"str"* e as informações de localização de cada ocorrência de cada *"str"* digitada. Após cada impressão relativa a uma *"str"* digitada é efetuada a limpeza da célula em que *"str"* estava sendo referida e ao final de todas as impressões é efetuada a limpeza das células da lista primária.

3. Funções primárias

3.1) Função "word()":

- Entrada: *"char *texto"* – O texto digitado.
- Saída: Retorna o número de palavras que há no texto.

3.2) Função "line()":

- Entrada: *"char *texto"* – O texto digitado.
- Saída: Retorna o número de linhas que há no texto.

3.3) Função "parag()":

- Entrada: *"char *texto"* – O texto digitado.
- Saída: Retorna o número de parágrafos que há no texto.

3.4) Função "primo()":

- Entrada: *"int n"* – O número de palavras a serem digitadas.
- Saída: Retorna 1 se "n" for primo e 0 caso "n" não for primo.

3.5) Função "ocorrencia()":

- Entrada: *"char *texto"* – O *"texto"* digitado.
  *"char *string"* – Palavra que desejamos saber se está contida no *"texto".*
- Saída: O número de vezes em que *"char *string"* está em *"char *texto".*

3.6) Função "apropriado()":

- Entrada: *"char *string"* – Palavra que desejamos saber se está contida no *"texto".*
  "int tam" – Tamanho de "*char *string".*
- Saida: Retorna o índice de "tab" apropriado para "*char *string*".

3.7) Função "pag()":

- Entrada: *"char *texto"* – O texto digitado.
  "*char *rst"* – A palavra digitada.

- Saída: Retorna quantos parágrafos tem entre o início do texto e *"rst".*

3.8) Função "col()" :

- Entrada: *"char *texto"* – O texto digitado.
  "*char *rst"* – A palavra digitada.

- Saída: Retorna em qual coluna está a palavra *"rst".*

3.9) Função "lin()" :

- Entrada: *"char *texto"* – O texto digitado.
  "*char *rst"* – A palavra digitada.

- Saída: Retorna quantas linhas tem entre o início do texto e *"rst"*.

3.10) Função "adiciona()" :

- Entrada: "*first *tab"* – Algum elemento de *"tab"* definido pela função ***"apropriado()"***.
  *"first *nova"* – A célula preenchida *"nova".*

- Saída: Apesar de a função não retornar um valor ela adiciona a célula *"nova"* ao índice correto de *"tab"* preenchendo um elemento da lista primária.

3.11) Função "adiciona2()" :

- Entrada: "*second *header"* – É a chave *"nova->inicio".*
  *"second *begin"* – A célula preenchida de cabeça "head".

- Saída: Apesar de a função não retornar um valor ela adiciona a célula *"begin"* a nova->inicio.

3.12) Função "tabv()":

- Entrada: "*first *tab[]"* – Vetor de ponteiros para lista primária.
  "int n" – Número de índices do vetor acima.

- Saída: Apesar de a função não retornar um valor ela cria todas as cabeças do vetor *"tab".*

3.13) Função "preencherF()":

- Entrada: *"first *nova"*– A célula a ser preenchida *"nova".*
  *"char *str"* – Palavra que será preenchida em *"nova->termo".*

"int ocor" – Número de ocorrências de *"str"*.
*"second *head"* – Cabeça da lista secundária.

- Saída: Apesar de a função não retornar um valor ela preenche a célula *"nova"* deixando ela pronta para ser adicionada ao índice correto de *"tab".*

3.14) Função "preencherS()":

- Entrada: *"second *begin"*– A célula a ser preenchida *"begin".*
  "int a" – Parágrafo em que *"char *str"* está.
  "int b" – Linha em que *"char *str"* está.
  "int c" – Coluna em que *"char *str"* está..

- Saída: Apesar de a função não retornar um valor ela preenche a célula *"begin"* deixando ela pronta para ser adicionada a *"nova->inicio".*

3.15) Função "free_s()":

- Entrada: *"second *ini"*– A célula a ser limpa *"begin".*

- Saída: Apesar de a função não retornar um valor ela limpa a célula *"begin"* após ela ser utilizada completamente.

3.16) Função "free_p()":

- Entrada: *"first *tab"*– A lista a ser limpa *"tab".*

- Saída: Apesar de a função não retornar um valor ela limpa a lista "tab", porém índice a índice.

3.17) Função "limpabuffer()":

- Entrada: Não há entrada.

- Saída: Não há saída, a função apenas limpa o buffer para não haver problemas na hora de digitar um char ou uma string.