Projeto II - identificação de prefixos e indexação de dicionários

Data de entrega: Friday, 30 Jun 2023, 23:59
Arquivos requeridos: main.cpp (Baixar)
Número máximo de arquivos: 10
Tipo de trabalho: Trabalho individual

Objetivo

Este trabalho consiste na construção e utilização de estrutura hierárquica denominada ***trie*** (do inglês "re**trie**val", sendo também conhecida como **árvore de prefixos** ou ainda **árvore digital**) para a indexação e recuperação eficiente de palavras em grandes arquivos de dicionários (mantidos em memória secundária). A implementação deverá resolver dois problemas (listados a seguir), e os resultados deverão ser formatados em saída padrão de tela de modo que possam ser automaticamente avaliados no VPL.

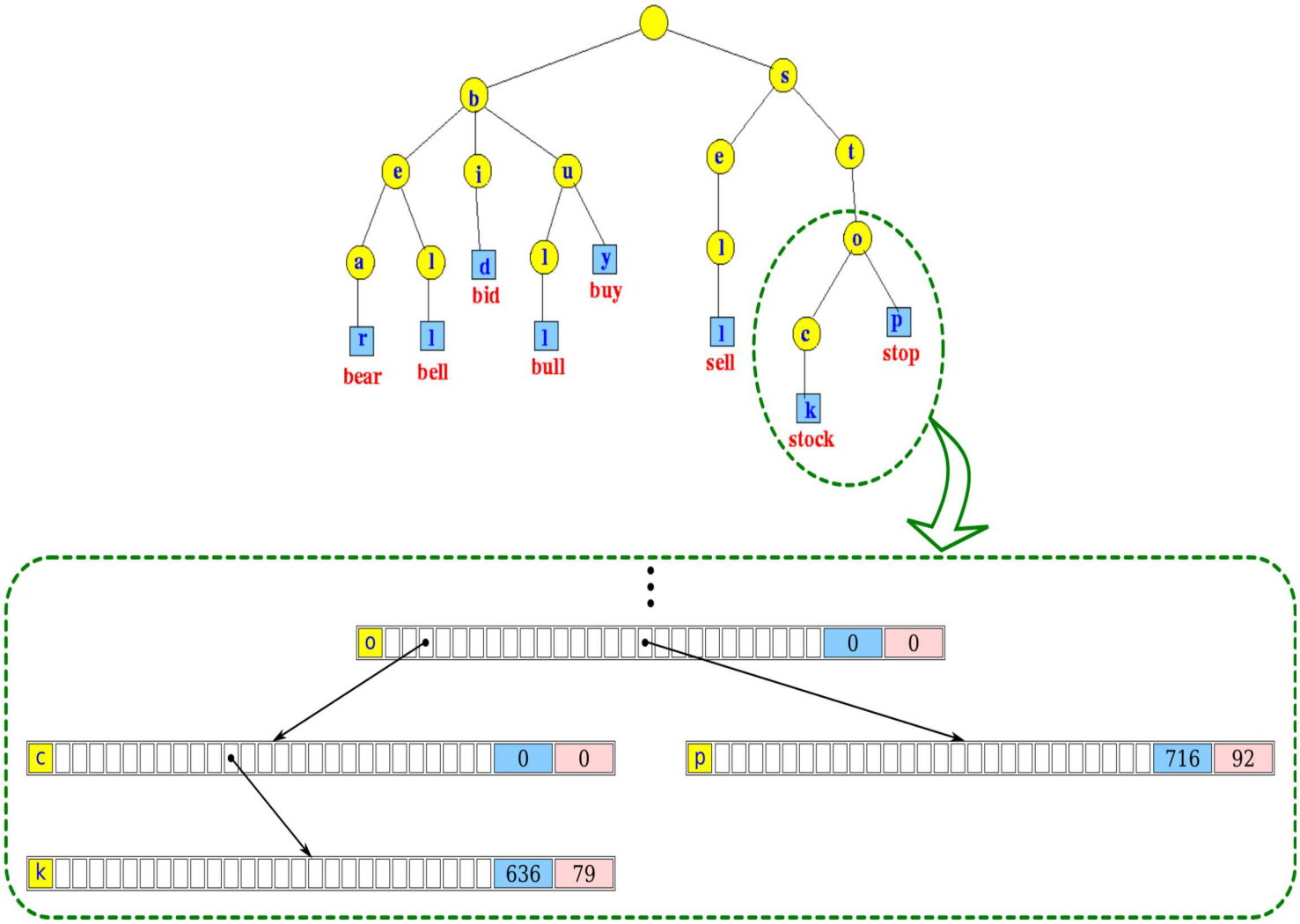
A figura a seguir exemplifica a organização de um arquivo de dicionário. Cada linha apresenta a definição de uma palavra, sendo composta, no início, pela própria palavra com todos os caracteres em minúsculo (somente entre 'a' (97) e 'z' (122) da tabela ASCII) e envolvida por colchetes, seguida pelo texto de seu significado. Não há símbolos especiais, acentuação, cedilha, etc, no arquivo.

dicionario1.dic

```
[bear]The definition of bear is a large mammal found in America and Eurasia which has thick fur or a big person or a person who is cranky and grumpy.
[bell]A hollow metal musical instrument, usually cup-shaped with a flared opening, that emits a metallic tone when struck.
[bid]The definition of bid means an offer of what someone will give for something.
[bull]The definition of a bull is an uncastrated male bovine animal, or is slang for nonsensical and untrue talk.
[buy]The definition of buy means to purchase or to get by exchange.
[sell]Sell is defined as to exchange something for money, act as a sales clerk or offer for sale.
[stock]The definition of stock is something that is in normal supply or common.
[stop]To stop is defined as to block, close, defeat, prevent from moving or bring to an end.
```

*esta linha (da palavra '**stop**') inicia pelo caracter **716** do arquivo e tem comprimento de **92** caracteres*

*esta linha (da palavra '**stock**') inicia pelo caracter **636** do arquivo e tem comprimento de **79** caracteres*



Materiais

De modo a exemplificar as entradas para o seu programa, seguem os arquivos de dicionário utilizados nos testes:

- dicionario1.dic
- dicionario2.dic

Dicas para implementação de *tries*:

- https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/tries.html
- https://towardsdatascience.com/implementing-a-trie-data-structure-in-python-in-less-than-100-lines-of-code-a877ea23c1a1
- https://www.geeksforgeeks.org/trie-insert-and-search/

Primeiro problema: identificação de prefixos

Construir a *trie*, em memória principal, a partir das palavras (definidas entre colchetes) de um arquivo de dicionário, conforme o exemplo acima. A partir deste ponto, a aplicação deverá receber uma série de palavras quaisquer (pertencentes ou não ao dicionário) e responder se trata de um prefixo (a mensagem '**P is prefix of N words**' deve ser produzida, onde **P** é o nome da palavra e **N** é a quantidade de palavras) ou não (a mensagem '**P is not prefix**' deve ser produzida na saída padrão). Sugestão de nó da *trie*:

```
NoTrie {
    char           letra;        //opcional
    NoTrie        *filhos[26];   //pode ser uma 'LinkedList' de ponteiros
    unsigned long  posição;
    unsigned long  comprimento;  //se maior que zero, indica último caracter de uma palavra
}
```

Segundo problema: indexação de arquivo de dicionário

A construção da *trie* deve considerar a localização da palavra no arquivo e o tamanho da linha que a define. Para isto, ao criar o nó correspondente ao último caracter da palavra, deve-se atribuir **a posição do caracter inicial** (incluindo o abre-colchetes '`[`'), seguida pelo **comprimento da linha** (não inclui o caracter de mudança de linha) na qual esta palavra foi definida no arquivo de dicionário. Caso a palavra recebida pela aplicação exista no dicionário, estes dois inteiros devem ser determinados. **Importante:** uma palavra existente no dicionário também pode ser prefixo de outra; neste caso, o caracter final da palavra será encontrado em um nó **não-folha** da *trie* e duas linhas deverão ser produzidas na saída padrão, a mensagem '**P is prefix of N words**' na primeira linha, e '**P is in (D,C)**' na linha seguinte (sendo **D** a posição, e **C** o comprimento).

Exemplo:

Segue uma entrada possível para a aplicação, exatamente como configurada no VPL, contendo o nome do arquivo de dicionário a ser considerado, cuja a *trie* deve ser construída (no caso para '`dicionario1.dic`' da figura acima), e uma sequência de palavras, separadas por um espaço em branco e finalizada por '0' (zero); e a saída que deve ser produzida neste caso.

- **Entrada:**

  ```
  dicionario1.dic bear bell bid bu bull buy but sell stock stop 0
  ```

- **Saída:**

```
bear is prefix of 1 words
bear is at (0,149)
bell is prefix of 1 words
bell is at (150,122)
bid is prefix of 1 words
bid is at (273,82)
bu is prefix of 2 words
bull is prefix of 1 words
bull is at (356,113)
buy is prefix of 1 words
buy is at (470,67)
but is not prefix
sell is prefix of 1 words
sell is at (538,97)
stock is prefix of 1 words
stock is at (636,79)
stop is prefix of 1 words
stop is at (716,92)
```