Trabalho de Implementação II - identificação de prefixos e indexação de dicionários

Disponível até: segunda, 3 Dez 2018, 23:55
Arquivos requeridos: main.cpp (Baixar)
Número máximo de arquivos: 10
Tipo de trabalho: Trabalho individual

Objetivo

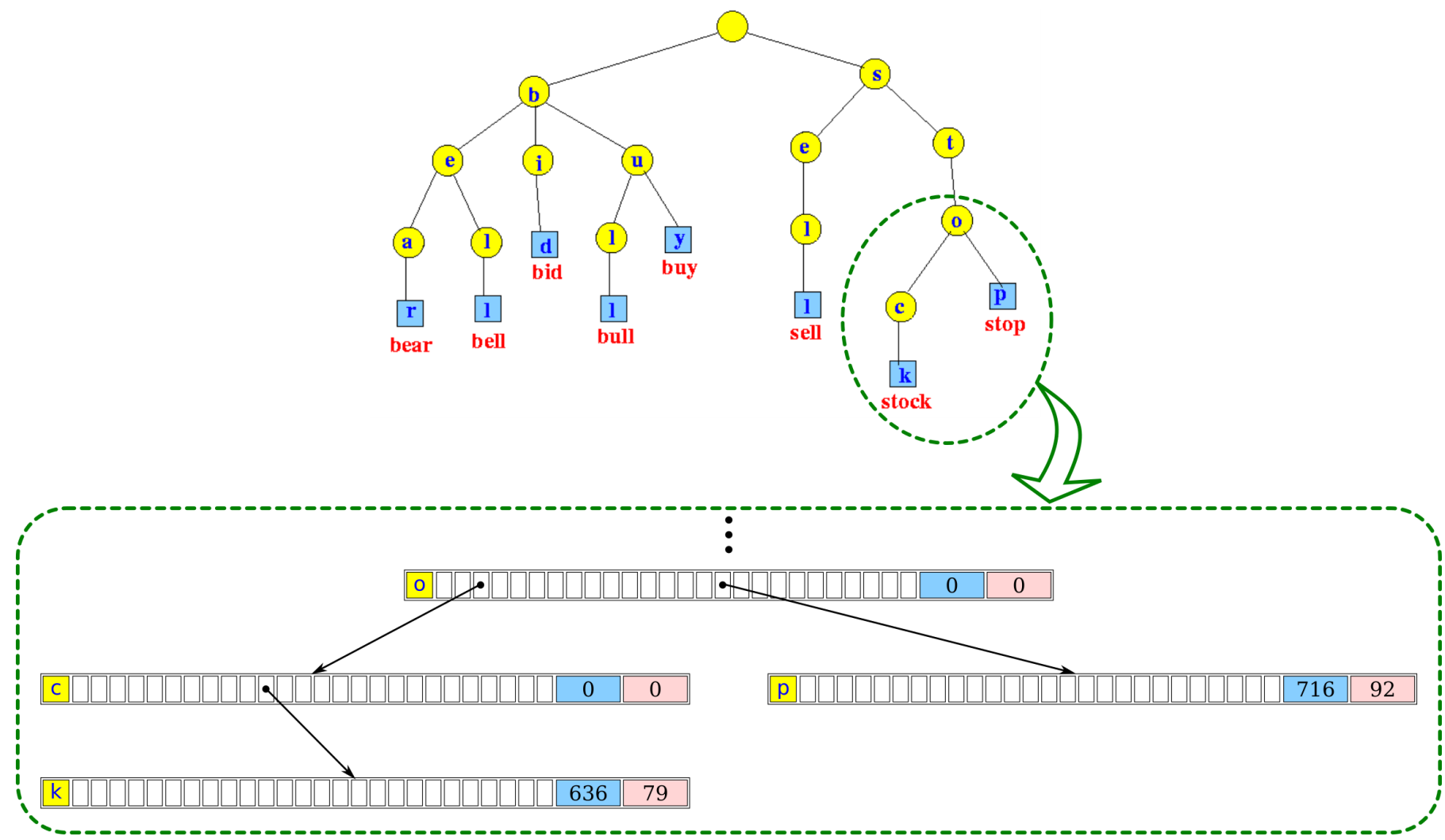
Este trabalho consiste na construção e utilização de estrutura hierárquica denominada ***trie*** (do inglês "re**trie**val", sendo também conhecida coo **árvore de prefixos** ou ainda **árvore digital**) para a indexação e recuperação eficiente de palavras em grandes arquivos de dicionários (mantidos 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 '**is prefix**' deve ser produzida) ou não (a mensagem '**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 contrução 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 produzidos. **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 também deve-se produzir os dois inteiros (posição e comprimento) na saída padrão.

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:**

```
0 149
150 122
273 82
is prefix
356 113
470 67
is not prefix
538 97
636 79
716 92
```

Entrega

- **Individual** ou **em dupla**
- Composição da nota:
  - Nota automática do VPL[(*)]: **70%**
    - [(*)] Caso algum caso de teste não tenha sido bem sucedido, o aluno **opcionalmente** poderá defender sua solução na **primeira semana de dezembro** reservado à **apresentação do Trabalho de Implementação II**
  - Relatório em PDF (utilize este link para a submissão) com todas as explicações pertinentes e documentação do código [(**)]: **30%**
    - [(**)] Sugere-se a escrita no próprio código usando a notação Doxygen com a geração automática de LaTeX/PDF