MAC323 EXERCÍCIO-PROGRAMA 3

BUSCA DE PALAVRAS

Y. KOHAYAKAWA

Data de entrega: 16/6/2008[1]

Introdução. Neste EP, você deverá implementar *árvores rubro-negras esquerdistas* (*left-leaning red-black trees*), versão de Maresias, como estudado em sala. Veja

`http://www.cs.princeton.edu/~rs/talks/LLRB/RedBlack.pdf`

Com tais árvores, você irá implementar uma *versão* de o que é conhecido como *índices invertidos*; você pode achar a seguinte entrada da Wikipedia interessante:

`http://en.wikipedia.org/wiki/Inverted_index`

Basicamente, você deverá escrever um programa que recebe como dados vários livros disponíveis no Projeto Gutenberg (digamos, algumas centenas de livros), e que produz um índice invertido para que, posteriormente, o usuário possa rapidamente localizar todas as ocorrência de qualquer palavra nesses livros.

Ademais, alimentando um dicionário ao seu programa, ele deve encontrar todas as palavras que ocorrem nesses livros que não estão no dicionário. Uma versão dessa parte do EP é o programa `wordtest` de Knuth (escrito em CWEB). Veja

`http://www.ime.usp.br/~yoshi/mac122/EPs/wordtest/`

Não é necessário você estudar a fundo a estrutura de dados conhecida como *treap*, usada em `wordtest`. Entretanto, recomendo fortemente que você leia `wordtest`, como fonte de inspiração.[2]

Especificações. Discutimos aqui detalhes relacionados à implementação e à interface com o usuário.

As tarefas principais. A sua tarefa tem basicamente quatro partes: (A) implementar uma ARNE (árvore rubro-negra esquerdista), (B) implementar um índice invertido usando sua ARNE, (C) escrever um módulo de interação com o usuário, através do qual ele poderá executar buscas de palavras (como fazemos no `google`), e (D) implementar um *filtro* (um 'verificador ortográfico'), que, dado um *dicionário*, verifica quais palavras que constam no índice invertido não ocorrem no dicionário.

Implementação da ARNE. Use suas notas de aula e o material disponibilizado por Sedgewick (note que ele disponibiza uma implementação em Java).

Data: Versão de 23 de junho de 2008 (20:04).

[1]Será praticamente impossível adiar a entrega deste EP, pois temos de fechar o semestre etc. O EP4 será opcional, para ajudar aqueles que quiserem aumentar a nota de EPs (todos poderão entregar). O enunciado do EP4 sairá antes da entrega do EP3, mas não muito antes.

[2]É muito fácil entender o que é um treap: leia o texto inicial do §9 (página 6) de `wordtest.pdf`.

O índice invertido. O usuário especificará um conjunto de arquivos texto (livros do Projeto Gutenberg, por exemplo, `alice30.txt` (Alice no País das Maravilhas, de Lewis Carroll)). *Você deve supor que o conjunto de arquivos pode ser armazenado em memória principal.* Usando sua ARNE, você deve implementar um índice invertido para o conjunto das palavras que ocorrem nesses livros. Para cada palavra, você deve armazenar onde ela ocorre em cada arquivo, de forma que, quando necessário, o programa poderá produzir o conjunto de todas as linhas de todos os arquivos em que a palavra ocorre. Seu programa deve ser capaz de produzir esse conjunto de linhas em tempo não muito maior que $O(t)$, onde t é o número total de vezes que a palavra ocorre no dado conjunto de arquivos.

Buscas que o usuário poderá fazer. O usuário executará seu programa com a linha de comando `ep3 alice30.txt` ou `ep3 -flista`. No primeiro caso, `alice30.txt` deverá ser usado para se montar o índice invertido. No segundo caso, supomos que `lista` é um arquivo que contém o nome dos arquivos que devem ser lidos para se montar o índice invertido (por simplicidade, suponha que os nomes dos arquivos em `livro` vêm um por linha).

Uma vez montado o índice invertido, seu programa deve entrar em um modo interativo, respondendo com um *prompt*:

```
ep3 >
```

Suponha que fornecemos `alice30.txt` como entrada. O usuário poderia dizer

```
ep3 > life
```

para receber a saída

```
  486:that it seemed quite dull and stupid for life to go on in the
  633:been to the seaside once in her life, and had come to the general
  737:them, as if she had known them all her life.  Indeed, she had
 1065:this sort of life!  I do wonder what CAN have happened to me!
 1382:      Has lasted the rest of my life.'
 1413:her life before, and she felt that she was losing her temper.
 1910:saw in my life!'
 2265:ever was at in all my life!'
 2450:curious croquet-ground in her life; it was all ridges and
 2584:at HIS time of life.
 2796:slowly after it:  'I never was so ordered about in all my life,
 3851:remembering her own child-life, and the happy summer days.
```

(os números indicam as linhas impressas). Na verdade, você deverá implementar uma 'calculadora RPN'[3] para manipular os conjuntos de linhas que o usuário criará com suas consultas: o usuário poderá calcular as linhas nas quais ocorre a palavra `life`, unida com o conjunto das linhas em que ocorre a palavra `nonsense`, obtendo um conjunto de linhas S. Da mesma forma, o usuário poderá calcular o conjunto T das linhas em que ocorrem a palavra `she` ou `Alice` (note que T será grande). O usuário poderia então calcular a interseção de S e T, e poderia pedir para ver tal conjunto. A saída seria algo como:

```
  737:them, as if she had known them all her life.  Indeed, she had
 1413:her life before, and she felt that she was losing her temper.
 2376:  'Nonsense!' said Alice, very loudly and decidedly, and the
 3415:  'Don't talk nonsense,' said Alice more boldly:  'you know
 3774:  'Stuff and nonsense!' said Alice loudly.  'The idea of having
```

Ao fazer tais operações, a unidade que consideramos foi a de *linha.* O usuário poderá expecificar contextos maiores: em vez de considerarmos linhas, o usuário poderá especificar que a unidade

[3]Veja `http://en.wikipedia.org/wiki/Reverse_Polish_notation`.

deve ser um conjunto de ℓ linhas consecutivas. Chamemos tais conjuntos de ℓ linhas de *contexto*. Note que, se uma palavra ocorre no texto, há ℓ contextos que contêm essa palavra. Por exemplo, considerando $\ell = 3$, a palavra `lazy` ocorre nos seguintes três contextos:

```
2776:  They very soon came upon a Gryphon, lying fast asleep in the
2777:sun.  (IF you don't know what a Gryphon is, look at the picture.)
2778:'Up, lazy thing!' said the Queen, 'and take this young lady to

2777:sun.  (IF you don't know what a Gryphon is, look at the picture.)
2778:'Up, lazy thing!' said the Queen, 'and take this young lady to
2779:see the Mock Turtle, and to hear his history.  I must go back and

2778:'Up, lazy thing!' said the Queen, 'and take this young lady to
2779:see the Mock Turtle, and to hear his history.  I must go back and
2780:see after some executions I have ordered'; and she walked off,
```

Por conveniência, se o usuário pedir para imprimir os contextos que contêm a palavra `lazy`, seu programa deve ter como saída

```
2776:  They very soon came upon a Gryphon, lying fast asleep in the
2777:sun.  (IF you don't know what a Gryphon is, look at the picture.)
2778:'Up, lazy thing!' said the Queen, 'and take this young lady to
2779:see the Mock Turtle, and to hear his history.  I must go back and
2780:see after some executions I have ordered'; and she walked off,
```

Isto é, seu programa deve imprimir as linhas nos contextos pedidos, sem repetição. Em geral, se o usuário pede para imprimir um dado conjunto de contextos, a saída deve ser tal que as linhas são impressas em ordem, sem repetição. No caso de estarmos tratando de vários livros, sua saída também tem de especificar os livros. Suponha que, além de `alice30.txt`, fornecemos o arquivo `wrnpc12.txt` (Guerra e Paz, de Tolstoy), e o usuário pede as ocorrências de `love`, a saída seria algo como ($\ell = 1$):

```
alice30.txt:
    526:                          (WITH ALICE'S LOVE).
   2652:'tis love, 'tis love, that makes the world go round!"'
wrnpc12.txt:
    974:does not love liberty and equality? Even our Saviour preached
   1415:of, and until you have ceased to love the woman of your choice and
   2226:"And she's in love with Boris already. Just fancy!" said the
   2355:"Natasha," he said, "you know that I love you, but..."
   [... etc ...]
```

O usuário deverá também ser capaz de adicionar e remover livros durante a interação com seu programa. Por exemplo, o usuário poderia ter executado seu programa com `ep3 alice30.txt`, mas posteriormente ele poderia querer adicionar `wrnpc12.txt` (a partir desse ponto, todas as buscas devem ser feitas nos dois arquivos). Mais tarde, ele poderia querer remover `alice30.txt`. O seu programa deverá permitir tais operações.

Observação. Os conjuntos que a sua calculadora manipula devem ser conjuntos de contextos. Suponha que $\ell = 3$ e que S é o conjunto de contextos em que ocorre a palavra `lazy` em `alice30.txt` (veja acima) e que T é o conjunto de contextos em que ocorre a palavra `Gryphon`. A interseção de S e T é constituída de dois contextos:

```
2776:  They very soon came upon a Gryphon, lying fast asleep in the
2777:sun.  (IF you don't know what a Gryphon is, look at the picture.)
2778:'Up, lazy thing!' said the Queen, 'and take this young lady to
```

```
2777:sun.  (IF you don't know what a Gryphon is, look at the picture.)
2778:'Up, lazy thing!' said the Queen, 'and take this young lady to
2779:see the Mock Turtle, and to hear his history.  I must go back and
```

Se o usuário pedir para imprimir $S \cap T$, a saída deverá ser

```
2776:  They very soon came upon a Gryphon, lying fast asleep in the
2777:sun.  (IF you don't know what a Gryphon is, look at the picture.)
2778:'Up, lazy thing!' said the Queen, 'and take this young lady to
2779:see the Mock Turtle, and to hear his history.  I must go back and
```

Especificações do modo interativo. Descrevemos aqui como o usuário irá se comunicar com seu programa no modo interativo. Já vimos que seu programa será executado de uma das seguintes formas: `ep3 <arquivo>` ou `ep3 -f<arquivo>`. No modo interativo, seu programa deve apresentar um *prompt*, por exemplo,

```
ep3 >
```

Ao digitar uma palavra (e `Enter`!):

```
ep3 > <palavra>
```

seu programa deve calcular o conjunto S dos contextos em que `<palavra>` ocorre.[4] Tal conjunto deve ser empilhado na pilha dos resultados de sua calculadora RPN. Dessa forma, o usuário cria vários conjuntos de contextos. O usuário pode operar com esses contextos, com os operadores `+`, `-`, e `\`. Ao receber `+`,

```
ep3 > +
```

sua calculadora deve desempilhar os dois conjuntos S e T no topo da pilha, deve computar $S \cup T$, e deve empilhar $S \cup T$ (o conjunto $S \cup T$ não deve ser impresso). Os operadores `-` e `\` são semelhantes: eles servem para o usuário computar $S \cap T$ e $S \setminus T$. O usuário poderá também pedir para ver o conjunto S que está no topo da pilha:

```
ep3 > .p
```

Note que o caracter `.` serve para dizer ao seu programa que `p` é um comando. Ao receber `.p`, seu programa deve imprimir os contextos do conjunto que está atualmente no topo da pilha; o conjunto deve continuar na pilha. Para remover o conjunto no topo da pilha, o usuário dirá

```
ep3 > .D
```

Para adicionar um novo arquivo aos dados, o usuário dirá

```
ep3 > .+<nome do arquivo>
```

Para remover um arquivo do conjunto de dados, o usuário dirá

```
ep3 > .-<nome do arquivo>
```

Às vezes, é útil trocar a ordem dos dois elementos no topo da pilha (isto é, se S está no topo e T está logo abaixo dele, após essa troca ficamos com T no topo e S logo abaixo). Para executar essa troca, o usuário dirá

```
ep3 > .S
```

Uma outra operação de rearranjo da pilha (que não é exatamente uma operação de pilha), mas que é muito útil em calculadoras RPN, é uma operação de *rotação*: se os elementos na pilha são, em ordem, a partir do topo, $S_1, \ldots, S_t$, após essa operação, a pilha conterá, em ordem, a partir do topo, $S_2, \ldots, S_t, S_1$. Para executar essa operação, o usuário dirá

```
ep3 > .R
```

Para saber quantos elementos estão na pilha, o usuário dirá

```
ep3 > .NP
```

[4] Para nós, uma *palavra* é uma cadeia maximal de letras, isto é, uma seqüência contígua com elementos em `A..Za..z` que não está contida em uma tal cadeia maior.

Para saber quantos contextos contém o conjunto no topo da pilha, o usuário dirá

```
ep3 > .NT
```

Para definir o contexto (o parâmetro ℓ acima), o usuário dirá

```
ep3 > .<inteiro>
```

Com isso, seu programa deve entender que $\ell =$ `<inteiro>`. O valor *default* de ℓ é 1 (isto é, os contextos são, inicialmente, linhas).

Seu programa deve calcular quanto tempo ele leva para fazer cada uma das operações pedidas pelo usuário. Assim, seu programa deve imprimir, depois de cada execução das operações `+`, `-` e `\`, o tempo que ele gastou para calcular o conjunto resposta. Ademais, quando o usuário pede que seu programa faça a busca de `<palavra>`, seu programa deve também imprimir o tempo que ele precisou para fazer isso (o `google` diz quanto tempo ele leva para executar uma busca). Se o usuário quiser 'desligar' essa característica de seu programa, ele dirá

```
ep3 > .q
```

O usuário poderá querer saber qual é o conjunto de linhas que os dados contêm. Para criar tal conjunto "universo" (com os dados daquele momento), ele dirá

```
ep3 > .U
```

Para saber quais linhas não contém a palavra `the`, ele pode fazer

```
ep3 > .U
ep3 > the
ep3 > \
ep3 > .p
```

Para que o usuário possa executar um 'programa' com os comandos que descrevemos acima, seu EP deverá aceitar a opção `-b`. Quando seu EP é executado com a chamada `ep3 -b -f<arquivo>`, ele deve receber no *stdin* uma seqüência de comandos (do modo interativo, como descrito acima) e ele deve executá-las na ordem (você pode supor que os comandos vêm um por linha). O resultado dessa computação deve ser enviado para o *stdout*. Note que, como não há interação nesse caso, seu programa não deve imprimir nenhum prompt. Por exemplo, se o usuário diz

```
genio@porsche:~$ ep3 -b alice30.txt < sem_the_a
```

e o arquivo `sem_the_a` contém

```
.U
the
a
+
\
.NT
```

o resultado será o número de linhas de `alice30.txt` que não contêm nenhuma das palavras `the` e `a`.

O verificador ortográfico. Você deverá basicamente reimplementar o `wordtest` nessa parte de seu EP. Você poderá comparar a eficiência de seu programa com a eficiência do `wordtest` (talvez o ponto é que o *monitor* poderá fazer isso!).

Suponha que `dicio.txt` contém um conjunto de palavras, ordenadas de forma crescente na ordem lexicográfica. O usuário pode fornecer esse arquivo como um *dicionário* para o seu programa da seguinte forma:

```
ep3 > .Wdicio.txt
```

Feito isso, ao dizer

```
ep3 > .wordtest
```

seu programa deve produzir um conjunto de palavras W contendo as palavras que ocorrem nos livros atualmente sendo considerados e que não estão no arquivo `dicio.txt`. Por simplicidade, essas operações não devem afetar a pilha de sua calculadora RPN. Para imprimir W, o usuário dirá

```
ep3 > .PW
```

Na impressão de W, as palavras devem vir em ordem lexicográfica crescente. *Curiosidade*: para ordenar um arquivo de palavras `arq.txt`, o usuário poderá executar seu programa da seguinte forma:

```
genio@porsche:~$ ep3 -b -farq.txt < ordene > arq_ord.txt
```

onde `ordene` contém as instruções

```
.q
.Wdicio.txt
.wordtest
.PW
```

onde `dicio.txt` não contém nenhuma palavra.

Témino da execução. Para encerrar a execução do programa, o usuário dirá

```
ep3 > .quit
```

Observações

1. *Este EP é estritamente individual.* Programas semelhantes receberão nota 0.
2. Seja cuidadoso com sua programação (correção, documentação, apresentação, clareza do código, etc), dando especial atenção a suas estruturas de dados. A correção será feita levando isso em conta.
3. Comparem entre vocês o desempenho de seus programas.
4. Entregue seu EP através do sistema Paca.
5. Não deixe de incluir em seu código um *relatório* para discutir seu EP: discuta as estruturas de dados usadas, os algoritmos usados, etc. *Se você escrever claramente como funciona seu EP, o monitor terá pouca dificuldade em corrigi-lo, e assim você terá uma nota mais alta.* (Se o monitor sofrer para entender seu código, você pode imaginar o humor dele ao atribuir sua nota.)
6. Lembre que o monitor colocou no fórum recomendações sobre os EPs. Não deixe de segui-las: ele é quem decide sua nota!

Observação final. Enviem dúvidas para a lista de discussão da disciplina.

Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508–090 São Paulo, SP

Endereço Eletrônico: `yoshi@ime.usp.br`