MAC323 EXERCÍCIO-PROGRAMA DE RECUPERAÇÃO

GERAÇÃO DE TEXTOS ALEATÓRIOS

Y. KOHAYAKAWA

Data de entrega: 21/7/2008

Introdução. Neste EP, você deverá usar *árvores de sufixos*, para implementar uma solução alternativa para o problema da geração de textos aleatórios, discutido na aula de 3/7/2008.

Recomendo fortemente que você leia o Capítulo 3 de Kernighan e Pike, *The practice of programming* antes de começar a fazer este EP. Além disso, você deve ler sobre árvores de sufixos: leia as Seções 12.7 e 15.5 de Sedgewick (será também importante a Seção 15.4—veja também o Exercício 15.80). Para nós, é suficiente entendermos uma árvore de sufixos de um texto como sendo basicamente uma árvore de busca em que armazenamos todos os sufixos do texto.

O algoritmo alternativo. O algoritmo visto em sala é como segue. Temos como entrada um texto T. Queremos gerar um texto aleatório usando T.

Algoritmo Markov (NPREF = 2)

w_1, $w_2 \leftarrow$ as duas primeiras palavras do texto T
imprima w_1 e w_2
repita:
    $w_3 \leftarrow$ um dos sucessores do prefixo w_1 w_2 em T (escolha aleatória)
    imprima w_3
    $w_1 \leftarrow w_2$; $w_2 \leftarrow w_3$

O algoritmo alternativo que você deverá implementar é como segue. Suponha que D seja uma distribuição de probabilidade sobre os inteiros não negativos; se X tem distribuição D, escrevemos $\mathbb{P}_D(X = k)$ ou simplesmente $D(X = k)$ para a probabilidade de X assumir o valor k. Suporemos sempre que D tem suporte finito: $D(X = k)$ é nulos para todo $k \geq k_0$, onde k_0 é uma constante associada a D.

Algoritmo Markov Variável

$w_1, w_2, \ldots \leftarrow$ as k_0 primeiras palavras do texto T
imprima $w_1, \ldots, w_{k_0}$
repita:
    gere um inteiro k de acordo com a distribuição D
    $u_0, \ldots, u_{k-1} \leftarrow$ as últimas k palavras geradas
    $u_k \leftarrow$ um dos sucessores do prefixo $u_0 \ldots u_{k-1}$ em T (escolha aleatória)
        [caso $u_0 \ldots u_{k-1}$ não ocorra em T, use $u_l \ldots u_{k-1}$ com menor l possível]
    imprima u_k

Data: Versão de 8 de julho de 2008 (16:42). A menos de correções e esclarecimentos, esta é a versão final do enunciado.

Árvores de sufixos. Em nosso algoritmo, precisamos descobrir as ocorrências de $u_0 \dots u_{k-1}$ em T. Para fazer isso rapidamente, você *deve* implementar uma estrutura de dados adequada; você não pode percorrer todo o T toda vez que você quer encontrar as ocorrências de uma seqüência de palavras (como $u_0, \dots, u_{k-1}$). *Sugestão*: Monte uma árvore de sufixos para T. *Exemplo*: suponha que o texto é

```
as armas e os barões assinalados
```

Considere os sufixos[1]

```
as armas e os barões assinalados
armas e os barões assinalados
e os barões assinalados
os barões assinalados
barões assinalados
assinalados
```

Coloque os 6 sufixos acima em uma *ternary search trie* (TST). Como você pode localizar a seqüência de palavras `os barões` usando essa TST?

Alguns detalhes. Você deve supor que seu EP será testado tanto para textos pequenos como para textos bastante extensos (digamos, concatenações de uma dezena de livros). Para verificar a eficiência de seu EP, vamos gerar textos também bastante grandes ($\sim 10^6$ palavras).

Entradas e opções. A distribuição D será dada da seguinte forma: o inteiro k_0 será dado inicialmente, e então será dada a seqüência $d_0, \dots, d_{k_0-1}$, onde d_k é a probabilidade de uma variável aleatória X de distribuição D assumir o valor k (assim, $d_k = D(X = k)$ para todo $0 \leq k < k_0$). A partir daí, o texto T será dado. Por exemplo, seu programa poderia ser executado assim:

```
genio@porsche:~$ eprec < dt.in
```

onde o arquivo `dt.in` contém

```
3 0.0 0.5 0.5
as armas e os barões assinalados
```

Seu programa também deve admitir algumas opções de linha de comando. Por exemplo, o usuário poderá dar a semente 2008 para o gerador de números aleatórios, da sequinte forma:

```
genio@porsche:~$ eprec -s2008
```

(Se o usuário não der uma semente através da opção `-s`, use um valor *default*, digamos, o valor 0.) Se você quiser que seu programa tenha portabilidade (e que ele tenha uma boa nota!), você pode usar o gerador de números aleatórios em

```
http://www.ime.usp.br/~yoshi/2008i/mac323/EPs/EPRec/gb_flip
```

(Leia em `gb_flip.pdf` como esse módulo do *Stanford GraphBase* de Knuth pode ser usado—não é necessário voce entender o algoritmo que é implementado! Note que, por exemplo, `gb_flip` contém a função conveniente `gb_unif_rand()`.)

Para depuração (e correção de seu EP), implemente também a seguinte opção: se o usuário disser, digamos,

```
genio@porsche:~$ eprec -d2
```

[1] Será conveniente considerar apenas os sufixos cujos inícios coincidem com inícios de palavras.

seu programa deve imprimir todos os pares de palavras u_0 u_1 que ocorrem no texto, seguido da lista das palavras que ocorrem logo após esses pares (seu programa não deve gerar o texto aleatório). Por exemplo, para o texto[2]

```
Show your flowcharts and conceal your tables and I will be mystified.
Show your tables and your flowcharts will be obvious.
```

a saída deve ser

```
I will: be |
Show your: tables | flowcharts |
and I: will |
and conceal: your |
and your: flowcharts |
be mystified.: Show |
be obvious.:
conceal your: tables |
flowcharts and: conceal |
flowcharts will: be |
mystified. Show: your |
tables and: your | I |
will be: obvious. | mystified. |
your flowcharts: will | and |
your tables: and | and |
```

(note que queremos a saída em ordem alfabética). Acima, consideramos pares de palavras porque o parâmetro de `-d` foi 2. O usuário poderá indicar qualquer inteiro não negativo k. Para $k = 0$, sua saída deve ser (basicamente falando) uma lista das palavras no texto, pois consideramos que a seqüência vazia de palavras *ocorre em todo ponto do texto*.

Elaboremos o caso $k = 0$ um pouco: no algoritmo que você implementará, caso o inteiro aleatório k gerado for 0, então você deve escolher para u_k alguma palavra do texto, com todas as palavras do texto equiprováveis.

A opção `-t` será útil na seguinte situação: podemos não estar interessados em pares de palavras que ocorram uma única vez no texto. Para tratar desse caso, seu EP deve aceitar um 'limiar' através da opção `-t`: se o usuário disser

```
genio@porsche:~$ eprec -d2 -t1 < Myth.txt
```

onde `Myth.txt` contém o texto do *Mythical Man Month* acima, a saída deve ser

```
Show your: tables | flowcharts |
tables and: your | I |
will be: obvious. | mystified. |
your flowcharts: will | and |
your tables: and | and |
```

Seu programa deve imprimir apenas aqueles pares (*pares*, pois o usuário disse `-d2`) que ocorrem mais de uma vez no texto (pois o limiar dado pelo usuário foi 1). O usuário poderá dar qualquer inteiro não-negativo ℓ

[2]de F.P. Brooks, *The Mythical Man Month*

através da opção `-t`; seu programa deverá então considerar prefixos que ocorrem mais de ℓ vezes no texto de entrada.

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`