Segundo trabalho prático - MATA54

Gabriel Dahia

September 2, 2017

1 Considerações gerais

Este documento descreve as estruturas de dados e algoritmos utilizados para a implementação do segundo trabalho prático da disciplina MATA54 - Estrutura de Dados e Algoritmos II da Universidade Federal da Bahia.

Durante todo o texto, a não ser que indicado, n se refere ao número total de palavras na base de palavras e m se refere ao maior comprimento possível de uma palavra. Quando s é a palavra da operação de digitação de número i e p é a palavra da operação de digitação número $i + 1$, é dito que p *sucede* s.

Foram feitas as seguintes suposições sobre o funcionamento do sistema:

- Caso uma palavra esteja no banco de palavras, ela não será reinserida através de uma operação de inserção;
- O término de uma execução do programa significa o fim do texto digitado. Isto é, a primeira palavra digitada em uma execução do programa não sucede a última palavra digitada na execução anterior;
- Se uma palavra p é prefixo de outra palavra s, então $p < s$, onde $<$ é a ordem alfabética;
- Não há limite para o número de arquivos criados.

1.1 Trie

Tries são implementadas da seguinte maneira: cada nó possui uma lista de índices e uma lista de pares $< c, u >$, onde c é um caractere e u é um nó de

trie. Se em um nó v há um par $< c, u >$, então existe a transição de v para u mediante a leitura de c. No contexto de tries, r_T é a raiz da trie T – quando claro a partir do contexto, será escrito apenas r.

Uma consulta a uma trie T é representada por $T(r, p)$, p palavra, e tem seu significado definido de maneira recursiva: $T(v, \epsilon)$ corresponde a lista de índices de v, e $T(v, aw)$ é $T(u, w)$, se em v há o par $< a, u >$, e $\emptyset$, caso contrário.

Inserir o par $< p, i >$ em uma trie T, onde p é uma palavra e i é um índice, corresponde a criar as estruturas necessárias para que, se antes dessa inserção $T(r, p) = L$, após ela $T(r, p) = L \cup i$.

1.2 Manutenção de N chaves de maior frequência

Para manutenção das N chaves com maior frequência, escolheu-se por utilizar uma lista implementada em vetor com N pares chave-frequência.

Dadas chaves e suas frequências, mantém-se os N pares chave-frequência ordenados decrescentemente por valor de frequência. A operação de inserção (todas as referências à operação de inserção em listas desse tipo se referirão à operação aqui descrita) de um par chave-frequência pode ser feita $O(1)$ operações, desde que seja feita no fim do vetor subjacente. Para manter as chaves em ordem decrescente de frequência e manter apenas pares com valores de chave distintos, a lista é reordenada em $O(N)$ operações e, caso o número de pares supere N, o que apresenta menor frequência pode ser descartado em $O(1)$ operações. Portanto, a operação de inserção demanda $O(N)$ operações.

A vantagem em utilizar essa estrutura é que a sua implementação é extremamente simples, é possível acessar sequencialmente os pares de maneira linear e as constantes necessárias para realizar essas operações são pequenas. A desvantagem aparenta estar na complexidade linear na operação de inserção de um par.

Contudo, é importante perceber que o número de operações tem complexidade linear para o valor de N pares *guardados na lista*; ou seja, para um valor fixo de N, o custo por operação de inserção é constante. Apesar de esse mesmo argumento poder ser feito para qualquer estrutura de dados que guardasse exatamente N pares e descartasse os excedentes, como para esse sistema serão necessários apenas listas de tamanho 3, as vantagens da estrutura (acesso linear aos elementos ordenados, constantes por operação baixa e implementação ingênua) superam os benefícios de outras estruturas

de dados, como heaps ou árvores balanceadas.

2 Detalhes da implementação

Cada vez que uma palavra p é inserida na base de palavras, seja via uma operação de inserção, seja via uma operação de digitação, é atribuido a ela um identificador númerico inteiro i_p no intervalo $[0, n-1]$. Em particular, se antes da inserção de p existiam n palavras na base, $i_p \leftarrow n-1$ e, em seguida, o valor de n aumenta em uma unidade.

Os identificadores são atribuídos sequencialmente às palavras – se uma palavra p foi digitada antes de uma palavra s, então necessariamente $i_p < i_s$. Disso decorre que i_p identifica unicamente a palavra p, já que a operação de digitação não permite a reinserção de palavras na base e é um pressuposto que não serão inseridas palavras repetidas através da operação de inserção.

São mantidos, em memória principal, as n palavras do banco de palavras e suas frequências, duas tries, T e T', o índice da última palavra digitada e, para cada palavra, os índices e as frequências das três palavras que mais frequentemente a sucederam, guardados em listas como descritos na Seção 1.2 – os índices das palavras representam as chaves das listas.

2.1 Persistência

Mediante a inserção de p na base de palavras, são criados arquivos "i_p-*word.dat*" (doravante chamado de arquivo de p) e "i_p-*freq.dat*" (doravante chamado arquivo de frequência de p) que guardam, respectivamente, a palavra p e o número de vezes que p foi digitada. Quando uma palavra p sucede uma palavra s, caso essa seja a primeira ocorrência desse evento, é criado um arquivo "i_s-i_p-*freq.dat*" com valor 1 (esse arquivo será referido como arquivo de s-p).

Assim, toda vez que uma palavra p for digitada, sua frequência é lida a partir do seu arquivo de frequência, é incrementada e então é reescrita, agora atualizada, no dispositivo de memória secundária. Caso p tenha sucedido uma palavra s, é feita uma tentativa de leitura do arquivo de s-p. Se essa tentativa falhar, supõe-se que o arquivo não existe e, como visto anteriormente, o evento "p sucede s" ainda não havia ocorrido. Caso a tentativa seja bem sucedida, a frequência relativa é atualizada de maneira análoga a atualização da frequência absoluta.

A atribuição de um identificador único e a criação de arquivos individuais permite a atualização das frequências na memória secundária com operações que, descontando o tempo de acesso do arquivo, equivalem a acesso aleatório. Essa abordagem é superior a métodos de hashing extensível porque permite recuperar os dados acessando exatamente uma página.

No começo de cada execução do sistema, as palavras e seus identificadores são carregados na memória principal através da leitura sequencial dos n arquivos de palavra existentes. Quando da leitura do arquivo da palavra p, seu arquivo de frequência e, para todo $i_s < i_p$, os arquivos s-p e p-s serão também carregados na memória principal. Isso é suficiente para garantir a persistência dos dados.

2.2 Consulta de palavras

Para consultar se uma determinada palavra está ou não presente na base de palavras de maneira eficiente, é mantida uma trie T. Quando uma palavra p é inserida no banco de palavras ou carregada da memória secundária, inserimos $< p, i_p >$ em T. Para determinar se p é uma palavra correta, basta verificar se $T(r, p) \neq \emptyset$. Com isso, pode-se determinar se uma palavra está ou não presente no banco de palavras em complexidade $O(m)$.

2.3 Sugestão de próximas palavras

Uma vez digitada a palavra p, é possível obter as três palavras que mais frequentemente a sucederam até esse momento acessando a lista disponível em memória principal para esse fim. Nela, os elementos já se encontram apropriadamente ordenados. Essa consulta, então, pode ser respondida em $O(1)$ operações.

Para manter as listas de frequências relativas atualizadas, se p sucedeu s, como visto na Seção 2.1, a frequência relativa de s-p é recuperada da memória secundária (seja através da leitura bem ou mal sucedida do arquivo de s-p). De posse dessa frequência, é realizada uma operação de inserção na lista que mantém as frequências relativas de s. Pela estrutura da lista, isso é suficiente para que, na próxima requisição, todas listas apresentem a organização desejada. A lista, portanto, também pode ser atualizada em tempo constante.

2.4 Possíveis correções

Dada uma palavra p, as possíveis correções de p foram definidas como palavras que estão no banco de palavras, tem o mesmo comprimento de p e dela diferem pela substituição de um caractere. Definamos o conjunto

$$\Phi(p) \;=\; \{uw \;\mid\; \exists\, a \text{ caractere},\; uaw \;=\; p\}$$

É direto perceber que uma palavra s é uma possível correção de p se, e somente se, $\Phi(s) \cap \Phi(p) \neq \emptyset$.

De posse dessa observação, utiliza-se uma segunda trie T' para responder eficientemente às consultas sobre possíveis correções. Toda vez que uma palavra p é inserida no banco de palavras ou carregada da memória secundária, inserimos cada palavra em $\Phi(p)$, acompanhada de i_p, em T'. Cada palavra em $\Phi(p)$ possui comprimento de, no máximo, $m - 1$ caracteres – inserí-las em T' pode ser feito com $O(m^2)$ operações. Para obter as possíveis correções de uma palavra p, então, computamos $\bigcup_{s \in \Phi(p)} T'(r, s)$ em $O(m^2)$. Pode-se inserir os pares índice-frequência em uma lista como a descrita na Seção 1.2 com $O(n)$ operações; os três elementos presentes na lista após todas as inserções são as correções desejadas. A complexidade para encontrar as 3 possíveis correções mais frequentes é, portanto, $O(m^2 + n)$.

2.5 Impressão de frequências

Para imprimir as palavras e suas frequências em ordem alfabética, percorremos a trie T em ordem infixa e guardamos, na ordem em que são vistos, os índices presentes nas listas de seus nós. O custo computacional dessa operação é igual ao número de nós na trie T: $O(nm)$. Esse procedimento coloca os nós em ordem alfabética na lista retornada. Vale lembrar que as frequências com que as palavras aparecem estão armazenados na memória principal.

A consulta de quais palavras mais sucederam uma dada palavra é trivialmente respondida pela lista mantida em memória principal.