Aula 08: 26/03/2019

Tópico

- ST em skip list

Leitura

Tabelas de símbolos (PF), Elementary Symbol Tables (S&W), slides (S&W)

Vídeo

Elementary Symbol Table, S&W

Tabela de Símbolos

Uma **tabela de símbolos** (= *symbol table*) é um ADT que consiste em um conjunto de itens, sendo cada item um par (`chave`, `valor` ou `key-value`), munido de duas operações fundamentais:

- `put()`, que insere um novo item no conjunto, e
- `get()`, que busca o valor associado a uma dada chave.

Convenções sobre TSs:

- não há chaves repetidas (as chaves são duas a duas distintas),
- `null` nunca é usado como chave,
- `null` nunca é usado como valor associado a uma chave.

API

`public class ST<Key,Value>`

`public class`	`ST`	
	`ST()`	cria uma tabela de símbolos vazia
`void`	`put(Key key, Value val)`	insere o item `(key,val)` nesta tabela
`Value`	`get(Key key)`	busca o valor associado a `key`
`boolean`	`isEmpty()`	esta tabela está vazia?
`boolean`	`contains(Key key)`	a chave `key` está nesta tabela?
`Iterable<Key>`	`keys()`	lista todas as chaves desta tabela

Como estimar o desempenho de uma implementação de TS?

Durante a execução de `get(k)` ou `put(k,v)`, uma chave da TS é tocada quando comparada com k. O consumo de tempo é proporcional ao *número de chaves tocadas*

Skip list

```
public class SkipListST {
    private static final int MAXLEVELS = 31; // we have at most 31 linked lists.
    private int lgN;   // number of actual levels, the levels are 0,1,...,lgN-1, lgN <= 31
    private Node first;
    private int n = 0; // number of (key,value) pairs in the table

    private class Node {
        private String key;
        private Integer val;
        private Node[] next;

        public Node(int levels) {
            this(null, null, levels);
        }

       public Node(String key, Integer val, int levels) {
            this.key  = key;
            this.val  = val;
            this.next =  new Node[levels];
        }
    }

    public SkipListST() {
        first = new Node(MAXLEVELS);
        last  = null;
    }

    public boolean contains(String key) {
        return getNode(key) != null;
    }

    public int size() {
        return n;
    }

    public boolean isEmpty() {
        return n == 0;
    }

    public Integer get(String key) {
        Node q = getNode(key);
        if (q != null) return q.val;
        return null;
    }

    public void put(String key, Integer val) {
        if (val == null) {
```

```
            delete(key);
            return;
        }
        Node[] stop = new Node[MAXLEVELS];
        Node p = first;
        for (int k = lgN-1; k >= 0; k--) {
            p = rank(key, p, k);
            Node q = p.next[k];
            if (q != null && q.key.equals(key)) {
                q.val = val;
                return;
            }
            stop[k] = p; // record the stop
        }


        // key is not in the ST, pair (key, val) must be inserted int the ST
        // create a new node
        int levels = randLevel();
        Node newNode = new Node(key, val, levels);

        // update the number o levels
        if (levels == lgN+1) {
            first.next[lgN] = newNode;
            lgN++;
            levels--;
        }

        // insert the new node into 0,1,...,levels-1 linked lists
        for (int k = levels-1; k >= 0; k--) {
            Node temp = stop[k].next[k];
            stop[k].next[k] = newNode;
            newNode.next[k] = temp;
        }

        n++;
    }


    /** Remove key (and the corresponding value) from this symbol table.
     * If key is not in the table, do nothing.
     */
    public void delete(String key)  {
        Node[] stop = new Node[MAXLEVELS];
        Node   p = first;
        for (int k = lgN-1; k >= 0; k--) {
            p = rank(key, p, k);
            stop[k] = p; // record the stop
        }
```

```
    // q points to a node that possibly contains ley
    Node q = stop[0].next[0];

    // key is not in the ST
    if (q == null || q.key.equals(key)) return;

    // key is in the ST
    int levels = q.next.length;

    // delete q from the linked lists 0, 1, 2, ..., levels-1;
    for (int k = 0; k < levels; k++)
        stop[k].next[k] = q.next[k];

    // delete q from the linked lists 0, 1, 2, ..., levels-1;
    for (int k = lgN-1; k >= 0 && first.next[k] == null; k--) lgN--;

    // update last
    if (stop[0].next[0]  == null) last = stop[0];

    n--;
}

/**
 *  Returns uniformly at random an integer in 1,2,...,MAXLEVELS
 */
private int randLevel() {
    int level = 0;
    int r = StdRandom.uniform((1 << (MAXLEVELS-1)));
    while ((r & 1) == 1) {
        if (level == lgN) {
            if (lgN == MAXLEVELS) return MAXLEVELS;
            else return lgN + 1;
        }
        level++;
        r >>= 1;
    }
    return level+1;
}


private Node rank(String key, Node start, int k) {
    Node p = start;
    Node q = p.next[k];
    while (q != null && q.key.compareTo(key) < 0) {
        p = q;
        q = q.next[k];
    }
    return p;
}
```

```
}
```

Análise

Fato. O número esperado de níveis de uma *skip list* com n itens é no máximo $\lg n + 2$.

Fato. O número esperado de nós em uma *skip list* com n itens é $2n + O(\lg n)$.

Fato. O comprimento esperado de um caminho de busca em uma *skip list* com n itens é $2 \lg n + O(1)$.

Fato. Seja T o número de vezes que uma moeda é jogada até obtermos coroa. $E[T] = 2$.

Rascunho de prova. Considere a variável aleatória indicadora I_i tal que

- $I_i = 0$ se a moeda é jogada $< i$ vezes
- $I_i = 1$ se a moeda é jogada $\geq i$ vezes

Temos que $T = I_1 + I_2 + I_3 + \cdots$ e $E[I_i] = Pr[I_i] = 1/2^{i-1}$. Logo

$$E[T] = E[\sum_{i=1}^{\infty} I_i] = \sum_{i=1}^{\infty} E[I_i] = 1 + 1/2 + 1/4 + \cdots < 2.$$

A probabilidade de um nó ter `next.length` maior ou igual a i é a probabilidade de obtermos $i - 1$ caras nas primeiras jogadas da moeda que é $1/2^{i-1}$.

Seja `h` o maior valor tal que em uma *skip list* com `n` itens a lista `first.next[h]` tem algum item (`!= null`). Temos que $Pr[h \geq i] \leq n/2^{i-1}$. De fato,

$\Pr[h \geq i] = \Pr[\text{algum nó p está na lista i}] \leq \sum_p \Pr[\text{p está na lista i}] \leq n/2^{i-1}$

Logo,

$\Pr[h \geq c \lg n] \leq n/2^{c \lg n - 1} < n/2^{c \lg n} = n/n^c = 1/n^{c-1}$

Em palavras, `h` é $O(\lg n)$ com alta probabilidade. Se $n = 1000$ e $c = 3$ então a probabilidade do número de níveis ser maior que $3 \lg 1000 < 30$ é menor que 1 em um milhão.

Resultados experimentais

```
BinarySeachST> java Driver ../data/les-miserables.txt
Criando a ST com as palavras do arquivo '../data/les-miserables.txt' ...
ST criada em 1.532 segundos
ST contém 26764 itens

LinkedListST> java Driver ../data/les-miserables.txt
Criando a ST com as palavras do arquivo '../data/les-miserables.txt' ...
ST criada em 147.1 segundos
ST contém 26764 itens

SequencialListST> java Driver ../data/les-miserables.txt
Criando a ST com as palavras do arquivo '../data/les-miserables.txt' ...
ST criada em 115.227 segundos
ST contém 26764 itens
```

```
SkipListST> java Driver ../data/les-miserables.txt
Criando a ST com as palavras do arquivo '../data/les-miserables.txt' ...
ST criada em 1.09 segundos
ST contém 26764 itens
Início da consulta interativa. Tecle ctrl+D encerrar
>>>
```