MAC0323 – Algoritmos e Estruturas de Dados II

Bacharelado em Ciência da Computação

Primeiro Semestre de 2024

Prova Substitutiva – 2/7/2024

Nome completo: _____

NUSP: _____

Assinatura: _____

Instruções:

1. Não destaque as folhas deste caderno.
2. Preencha o cabeçalho acima. Sua assinatura atesta a autenticidade e originalidade de seu trabalho e que você se compromete a seguir o código de ética da USP em suas atividades acadêmicas, incluindo esta prova.
3. A prova pode ser feita a lápis. Cuidado com a legibilidade.
4. A prova consta de **3 questões**. Verifique antes de começar a prova se o seu caderno de questões está completo.
5. Não é permitido o uso de folhas avulsas para rascunho.
6. Não é permitido o uso de artefatos eletrônicos.
7. Não é permitida a consulta a livros, apontamentos ou colegas.
8. Não apague seus rascunhos. Eles podem ser considerados na correção.

DURAÇÃO DA PROVA: 2 horas

Questão	Nota
1	
2	
3	
Total	

Q1 (3.0 pontos) Considere a versão reduzida de LinearProbingHashST.java abaixo.

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.Queue;

public class LinearProbingHashST<Key, Value> {
    private static final int CAPACITY = 16; // very small table
    private int n;           // number of key-value pairs in the symbol table
    private int m;           // size of linear probing table
    private Key[] keys;      // the keys
    private Value[] vals;    // the values
    private int D;           // total displacement

    public LinearProbingHashST() {
        m = CAPACITY;
        n = 0;
        keys = (Key[])   new Object[m];
        vals = (Value[]) new Object[m];
    }

    // Bad hash function
    private int hash(Key key) {
        String s = (String)key;
        int h = 2 * ((s.charAt(0) - 'A') % 8);
        return h;
    }

    public void put(Key key, Value val) {
        int d = 0;
        int i = hash(key);
        while (keys[i] != null) {
            if (keys[i].equals(key)) {
                vals[i] = val;
                return;
            }
            i = (i + 1) % m;
            d++;
        }
        if (n == CAPACITY - 1) {
            System.err.println("Panic: table full");
            System.exit(-1);
        }
        keys[i] = key;
        vals[i] = val;
        n++;
        StdOut.println(key + " / " + hash(key) + " / " + d);
        D += d;
    }
```

```
    public int displacement() {
        return D;
    }

    public static void main(String[] args) {
        LinearProbingHashST<String, Integer> st
            = new LinearProbingHashST<String, Integer>();
        for (int i = 0; !StdIn.isEmpty(); i++) {
            String key = StdIn.readString();
            st.put(key, i);
        }
        StdOut.println("Total displacement: " + st.displacement());
    }
}
```

A função de hashing usada acima pode ser tabelada pelo programa abaixo:

```
import edu.princeton.cs.algs4.StdOut;

public class BadHash
{
    public static void main(String[] args)
    {
        for (char c = 'A'; c <= 'Z'; c++)
            StdOut.printf("%2c ", c);
        StdOut.println();
        for (char c = 'A'; c <= 'Z'; c++) {
            int h = 2 * ((c - 'A') % 8);
            StdOut.printf("%2d ", h);
        }
        StdOut.println();
    }
}
```

BadHash.java tem a seguinte saída:

```
$ java-algs4 BadHash
 A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
 0  2  4  6  8 10 12 14  0  2  4  6  8 10 12 14  0  2  4  6  8 10 12 14  0  2
$
```

(a) Diga qual será a saída de LinearProbingHashST.java quando executado com as primeiras 15 letras de seu nome. **Exemplo:** se seu nome fosse *Alan Mathison Turing*, você usaria a entrada

```
A L A N M A T H I S O N T U R
```

Observações. Ignore acentos. Caso seu nome não tenha 15 letras, adicione *von Neumann* no fim de seu nome. *Exemplo*: se seu nome fosse *Alan Turing*, você usaria a entrada

```
A L A N T U R I N G V O N N E
```

Não apague seu rascunho. Ele será usado para entender como você chegou a sua resposta.

Rascunho

Entrada e saída do programa
Entrada: Saída:

(b) Suponha que, depois da execução de LinearProbingHashST.java em (a), fazemos uma busca com sucesso na tabela resultante. Diga qual será o número esperado de posições do vetor keys que serão examinadas em tal busca (isto é, o número de *probes*), supondo que todas as chaves presentes têm a mesma probabilidade de ser buscada. Dito de outra forma, para cada elemento presente na tabela, ao buscar aquele elemento, fazemos um certo número de probes: quanto é a média aritmética de tais números de probes? Dê a resposta na forma A/B, com A e B inteiros.

Q2 (4.0 pontos) O seguinte programa implementa Quicksort de forma iterativa (sem recursão).

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.StdIn;
import edu.princeton.cs.algs4.Stack;

public class Q2 {

    private static int max;  // maximum stack size

    private static class Pair {
        int i, j;

        Pair(int a, int b) {
            i = a;
            j = b;
        }
    }

    public static void sort(int[] a) {
        int N = a.length;
        Stack<Pair> calls = new Stack<>();
        calls.push(new Pair(0, N - 1));
        max = 1;
```

```
        while (!calls.isEmpty()) {
            Pair p = calls.pop();
            if (p.i < p.j) {
                int j = partition(a, p.i, p.j);
                calls.push(new Pair(p.i, j - 1));
                calls.push(new Pair(j + 1, p.j));
                if (calls.size() > max) max = calls.size();
            }
        }
    }

    // Partition the subarray a[lo .. hi] so that
    // a[lo .. j - 1] <= a[j] <= a[j + 1 .. hi] and return the index j.
    private static int partition(int[] a, int lo, int hi) {
        int i = lo;
        int j = hi + 1;
        int v = a[lo];
        while (true) {
            while (a[++i] < v)
                if (i == hi) break;
            while (v < a[--j])
                if (j == lo) break; // redundant since a[lo] acts as sentinel
            if (i >= j) break;
            exch(a, i, j);
        }
        exch(a, lo, j);
        return j;
    }

    private static void exch(int[] a, int i, int j)
    {  int swap = a[i];  a[i] = a[j];  a[j] = swap;  }

    public static void main(String[] args) {
        int[] a = StdIn.readAllInts();
        sort(a);
        StdOut.println("Max stack size: " + max);
    }
}
```

(a) Seja $N \geq 1$. Suponha que executamos Q2.java dando como entrada os primeiros N naturais positivos em ordem: $1, 2, \ldots, N$. Diga qual será o tamanho máximo da pilha ou, equivalentemente, qual será a saída dessa execução? Justifique brevemente sua resposta, discutindo sucintamente a evolução do conteúdo da pilha.

(b) Suponha agora que substituímos o "while" de sort() em Q2.java pelo seguinte "while":

```
while (!calls.isEmpty()) {
    Pair p = calls.pop();
    if (p.i < p.j) {
        int j = partition(a, p.i, p.j);
        if (j - p.i <= p.j - j) {
            calls.push(new Pair(j + 1, p.j));
            calls.push(new Pair(p.i, j - 1));
        } else {
            calls.push(new Pair(p.i, j - 1));
            calls.push(new Pair(j + 1, p.j));
        }
        if (calls.size() > max) max = calls.size();
    }
}
```

Seja Q2b.java o programa que obtemos com essa substituição. Qual será a saída de Q2b.java quando executado como no item (a)? Justifique brevemente sua resposta, discutindo sucintamente a evolução do conteúdo da pilha.

(c) Suponha que executamos Q2b.java com uma entrada de N inteiros. Prove que, para qualquer tal entrada, em nenhum momento a pilha terá mais de $\log_2 N + 1$ elementos. Faça indução em N. (Procure escrever a asserção que você provará por indução cuidadosamente. Provavelmente a base da indução incluirá os casos $N = 1$ e $N = 2$.)

Q3 (4.0 pontos) Considere o seguinte programa:

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.In;
import edu.princeton.cs.algs4.ST;
import edu.princeton.cs.algs4.SET;

public class Q3
{
    public static void main(String[] args)
    {
        In in0 = new In(args[0]);
        In in1 = new In(args[1]);
        String query = args[2];
        boolean verbose = args.length > 3;

        TrieST<Integer> trie = new TrieST<>();
        while (!in0.isEmpty())
            trie.put(in0.readString(), 0);

        ST<Character, SET<Character>> code = new ST<>();
        while (in1.hasNextLine()) {
            String line = in1.readLine();
            if (line.equals("")) break;
            String[] tokens = line.split("\\s+");
            SET<Character> s = new SET<>();
            for (int i = 1; i < tokens.length; i++)
                s.add(tokens[i].charAt(0));
            code.put(tokens[0].charAt(0), s);
        }
```

```
        if (verbose) {
            StdOut.println("Dictionary size: " + trie.size());
            for (char c : code.keys())
                StdOut.println(c + ": " + code.get(c));
        }

        Iterable<String> s = trie.queryT9(query, code);
        StdOut.println(query + ": " + s);
    }
}
```

Suponha que o arquivo words.txt contém uma lista de "palavras válidas" (por exemplo, são as palavras de um dicionário). Suponha que o conteúdo de code.txt é

```
2 A B C a b c
3 D E F d e f
4 G H I g h i
5 J K L j k l
6 M N O m n o
7 P Q R S p q r s
8 T U V t u v
9 W X Y Z w x y z
```

Assim code.txt contém a "codificação T9": o dígito 2 pode valer qualquer uma das letras A, B, C, a, b, c, o dígito 3 pode valer qualquer uma das letras D, E, F, d, e, f, e assim por diante. Na sistema T9, podemos, por exemplo, especificar a palavra "tough" com "86844". Por outro lado, "669" pode especificar tanto a palavra "mow" como "now" (assim como "MOW" e "NOW"). Podemos usar o programa Q3.java para verificar quais palavras são especificadas por uma dada sequência de dígitos:

```
$ java-algs4 Q3 DATA/words.txt DATA/code.txt 86844
86844: tough
$ java-algs4 Q3 DATA/words.txt DATA/code.txt 669
669: mow now
$
```

O programa Q3.java recebe o nome de um arquivo com as palavras válidas e o nome do arquivo com o código T9, além da sequência de dígitos a ser "traduzida". A "tradução" em Q3.java ocorre na execução de trie.queryT9(), mas o método queryT9() não existe na implementação que vimos de tries. Sua tarefa nesta questão é implementar tal método: *mais especificamente, complete a versão estendida de Trie.java dada a seguir, implementando o método de assinatura*

```
    private void collectT9(Node x, StringBuilder prefix, String pattern,
                           Queue<String> results, ST<Character, SET<Character>> code)
```

de forma que Q3.java funcione como especificado acima. Naturalmente, a lista de palavras na "tradução" deve ser a lista de palavras codificadas pela sequência de dígitos que constam na lista dada de palavras válidas (o arquivo words.txt usado nos dois exemplos acima contém "tough", "mow" e "now", mas não contém "TOUGH", "MOW" e "NOW").

```
public class TrieST<Value> {
    private static final int R = 256;  // extended ASCII
    private Node root;                 // root of trie

    private static class Node {
        private Object val;
        private Node[] next = new Node[R];
    }

    [...]

    public Iterable<String> keysThatMatch(String pattern) {
        Queue<String> results = new Queue<String>();
        collect(root, new StringBuilder(), pattern, results);
        return results;
    }

    private void collect(Node x, StringBuilder prefix, String pattern,
                         Queue<String> results) {
        if (x == null) return;
        int d = prefix.length();
        if (d == pattern.length() && x.val != null)
            results.enqueue(prefix.toString());
        if (d == pattern.length()) return;
        char c = pattern.charAt(d);
        if (c == '.')
            for (char ch = 0; ch < R; ch++) {
                prefix.append(ch);
                collect(x.next[ch], prefix, pattern, results);
                prefix.deleteCharAt(prefix.length() - 1);
            }
        else {
            prefix.append(c);
            collect(x.next[c], prefix, pattern, results);
            prefix.deleteCharAt(prefix.length() - 1);
        }
    }

    /***************************************************************************
     * T9 methods
     ***************************************************************************/

    public Iterable<String> queryT9(String pattern,
                                    ST<Character, SET<Character>> code) {
        Queue<String> results = new Queue<String>();
        collectT9(root, new StringBuilder(), pattern, results, code);
        return results;
    }
```

```
    private void collectT9(Node x, StringBuilder prefix, String pattern,
                           Queue<String> results,
                           ST<Character, SET<Character>> code) {

        // Implemente este método

    }
}
```

Novamente: o que você deve fazer nesta questão é implementar o método collectT9() (não deixe de escrever a assinatura da função em sua solução). Note que, acima, trechos de Trie.java que podem ajudar em sua implementação de collectT9() são dados.

Importante. Sua implementação de collectT9() deve ter complexidade $O(T)$, onde T é o número de caracteres total na lista resultante de palavras (por exemplo, no exemplo "669" acima, $T = 6$).

Observações. Segue mais uma execução de Q3.java:

```
$ java-algs4 Q3 DATA/words.txt DATA/code.txt 5433 -
Dictionary size: 235886
2: { A, B, C, a, b, c }
3: { D, E, F, d, e, f }
4: { G, H, I, g, h, i }
5: { J, K, L, j, k, l }
6: { M, N, O, m, n, o }
7: { P, Q, R, S, p, q, r, s }
8: { T, U, V, t, u, v }
9: { W, X, Y, Z, w, x, y, z }
5433: lied life
$
```

Você pode achar conveniente lembrar a API de SET:

```
public class SET<Key extends Comparable<Key>> implements Iterable<Key>
-----------------------------------------------------------------------
    SET()                       ...  constructs an empty set
    void add(Key key)           ...  add key to set
    boolean contains(Key key)   ...  true iff key belongs to set
    int size()                  ...  number of elements in set
    boolean isEmpty()           ...  true iff set is empty
    String toString()           ...  human readable form of set
```

* * *

(Rascunho)