Programmieren 1: Testklausur 15.6.2015

Leibniz Universität Hannover

Human-Computer Interaction Group

Prof. Dr. Michael Rohs
michael.rohs@hci.uni-hannover.de

Aufgabe 1 (C)

Implementieren Sie die Funktion double **distSecondSmallestToSmallest(List *list)**.

Diese soll die absolute Differenz zwischen dem kleinsten Element und dem zweitkleinsten Element der Liste zurückgeben. Bei einer zu kurzen Liste soll der Wert 0 zurückgegeben werden.

Correct Output

```
line 112: check passed
line 113: check passed
line 114: check passed
line 115: check passed
line 116: check passed
line 117: check passed
line 118: check passed
line 119: check passed
line 120: check passed
```

List and Node Structures

```
typedef struct Node {
    double value;
    struct Node *next;
} Node;

typedef struct List {
    Node *first;
    Node *last;
} List;
```

- struct

  ```
  struct S {
      int i;
  };
  ```

  variables of type "struct S"

- typedef

  typedef <u>struct S</u> T;

  variables of type "T"

- typedef struct

  syntactic abbreviation

List Creation

```
List *newList(void) {
    return calloc(1, sizeof(List));
}

Node *newNode(double value) {
    Node *node = calloc(1, sizeof(Node));
    node->value = value;
    return node;
}
```

- calloc = allocate and clear (set each byte to 0)

List Printing

```
void print(List *list) {
    printf("[");
    for (Node *node = list->first; node != NULL; node = node->next) {
        printf("%g", node->value);
        if (node->next != NULL) {
            printf(" ");
        }
    }
    printf("]\n");
}
```

List Appending

```
void append(List *list, double value) {
    Node *p = newNode(value);
    if (list->last != NULL) list->last->next = p;
    list->last = p;
    if (list->first == NULL) list->first = p;
}
```

List from String

```
/*
 * Create a list from the given string.
 * Use "," (with surrounding whitespace) as the separator.
 * Example: toList("1, 2.5, -3.2, 49.1") --> [1.0 2.5 -3.2 49.1]
 */
List *toList(char *s) {
    ...
}
```

Testing: Compare Actual to Expected Values

```
int check_within(int line, double actual, double expected, double delta) {
    if (fabs(actual - expected) <= delta) {
        printf("line %d: check passed\n", line);
        return 1;
    } else {
        printf("line %d: actual value %g is not within %g "
                "of expected value %g.\n",
                line, actual, delta, expected);
        return 0;
    }
}
```

Testing

```
int tests(void) {
    ...
    check_within(__LINE__,                  // this line in the source code
        distSecondSmallestToSmallest(       // the function to test
            toList("1.5, 3.0")),            // creating the list [1.5, 3.0]
        1.5,                                // the expected value
        0.000001);                          // the allowed tolerance
    ...
    return 0;
}
```

produces either:
"line 123: check passed"
or
"line 123: actual value 0 is not within 1e-06 of expected value 1.5.

Main Function

```
int main(void) {
    tests();
    return EXIT_SUCCESS;
}
```

double distSecondSmallestToSmallest(List *list)

```
// return 0 if list is too short (no list or zero or one elements)
if (list == NULL || list->first == NULL || list->first->next == NULL) {
    return 0.0;
}
// assert: list has at least 2 elements
...
```

double distSecondSmallestToSmallest(List *list)

- Need to identify smallest (min1) and second smallest (min2) element (min2 ≥ m1)
- Result is min2 - min1
- Need to look at each element of list to decide which one is smallest (and which one is second smallest)
- Use a loop
  - New element might be smaller than smallest (or second smallest) seen so far
  - Update smallest and/or second smallest if necessary

double distSecondSmallestToSmallest(List *list)

```
double min1 = ?;
double min2 = ?;
...
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 and/or min2 if necessary...
}
return min2 - min1; // result is min2 - min1
```

Subproblem: Find Smallest Element

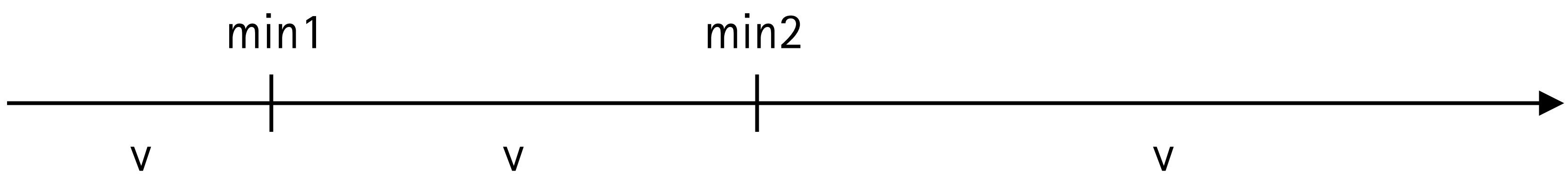
```
double min1 = ?;
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 if necessary...
    if (v < min1) {
        min1 = v;
    }
}
return min2 - min1; // result is min2 - min1
```

Subproblem: Find Smallest Element

```
double min1 = 0;
int min1valid = 0; // not valid yet
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 if necessary...
    if (v < min1 || !min1valid) {
        min1 = v;
        min1valid = 1; // min1 is now valid
    }
}
```

min1 and min2

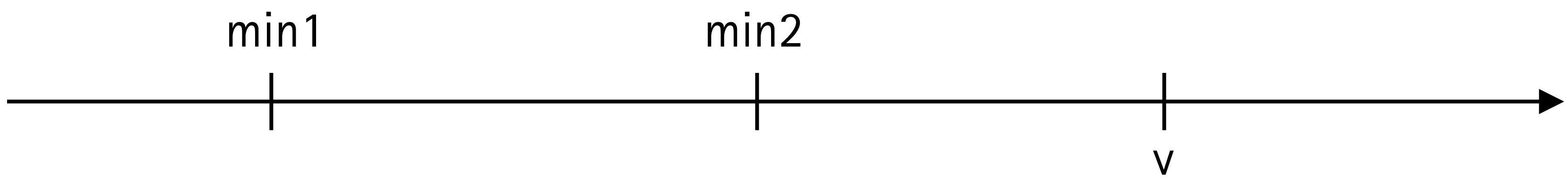
- min1: The smallest element seen so far
- min2: The second smallest element seen so far



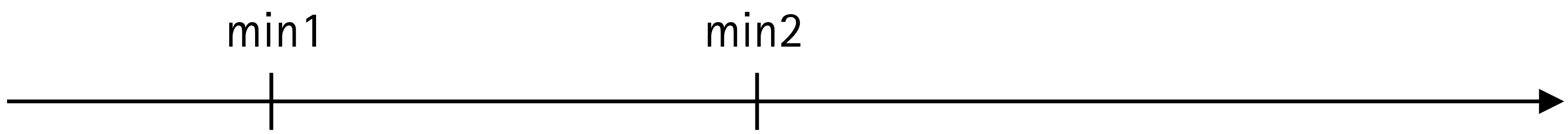
- 3 cases for next list element v
  - $v \geq min2$ → no change
  - $min1 \leq v < min2$ → min1 no change, $min2 = v$
  - $v < min1$ → $min1 = v$, $min2$ = old value of min1

min1 and min2

- min1: The smallest element seen so far
- min2: The second smallest element seen so far

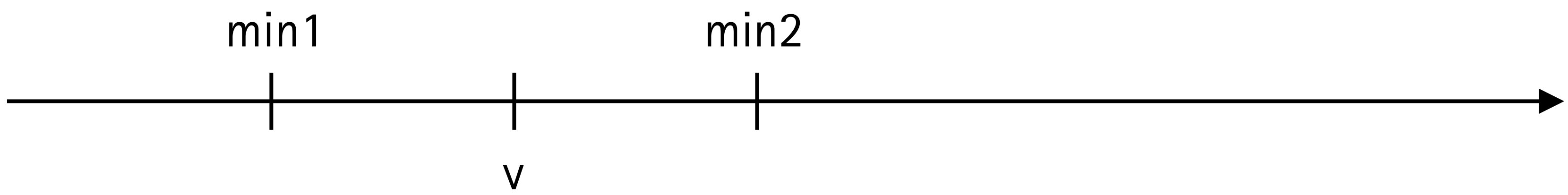


- Case 1: v ≥ min2 → no change



min1 and min2

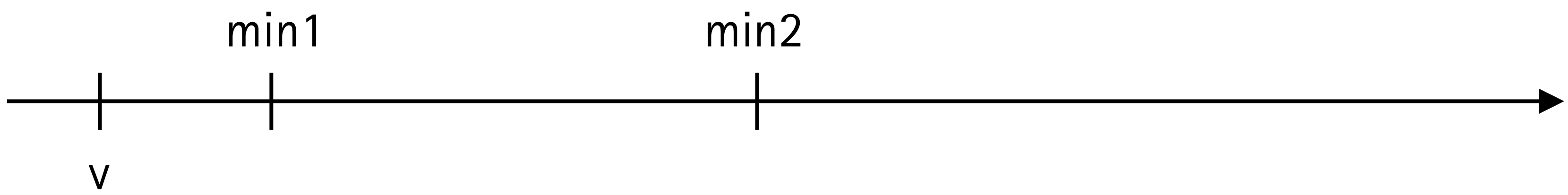
- min1: The smallest element seen so far
- min2: The second smallest element seen so far



- Case 2: min1 ≤ v < min2 → min1 no change, min2 = v

min1 and min2

- min1: The smallest element seen so far
- min2: The second smallest element seen so far



- Case 3: v < min1 → min1 = v, min2 = old value of min1

Find Smallest and Second Smallest Element

```
double min1 = 0;
int min1valid = 0; // not valid yet
double min2 = 0;
int min2valid = 0; // not valid yet
// need to look at each element
for (Node *node = list->first; node != NULL; node = node->next) {
    double v = node->value;
    // update min1 and/or min2 if necessary...
    ...
}
return min2 - min1; // result is min2 - min1
```

Find Smallest and Second Smallest Element

```
double v = node->value;
if (v < min1 || !min1valid) { // --v--min1--min2-->
    min2 = min1; // old value of min1
    min1 = v;
    min2valid = min1valid; // old value of min1valid
    min1valid = 1;
} else if (v < min2 || !min2valid) {// --min1--v--min2-->
    min2 = v;
    min2valid = 1;
}
```

Aufgabe 2a (Java)

a) Gegeben ist ein Suchbaum zum Speichern von ganzen Zahlen. Implementieren Sie die Methode void insert(int v) der Klasse Node, die den Wert v in den Suchbaum einfügt. Dabei soll die Suchbaumeigenschaft erhalten bleiben.

Correct Output

```
$ javac SearchTreeInsertion.java
$java SearchTreeInsertion
test passed
test passed
test passed
test passed
```

Search Tree

- Search tree criterion: For a node n with value v, all nodes in left subtree have values w with $w < v$ and all nodes in the right subtree have values w with $w \geq v$.
- Search tree insertion: If $w < v$ insert in n.left else insert in n.right. Special cases if n.left and/or n.right do not exist.

Search Tree Insertion

```
class Node {
    public int value;
    public Node left, right;
    public Node(Node l, int v, Node r) {
        this.left = l;
        this.value = v;
        this.right = r;
    }
    public Node(int v) {
        this.value = v;
        left = null;
        right = null;
    }
```

```
public void insert(int v) {
    if (v < value) {
        if (left != null) {
            left.insert(v);
        } else {
            left = new Node(v);
        }
    } else {
        if (right != null) {
            right.insert(v);
        } else {
            right = new Node(v);
}   }   }
```

Node Methods

```
public int depth() {
    if (left == null && right == null) {
        return 1;
    }
    else if (left != null && right == null) {
        return 1 + left.depth();
    }
    else if (left == null && right != null) {
        return 1 + right.depth();
    }
    else {
        return 1 + Math.max(left.depth(), right.depth());
    }
}
```

General approach: Look at all cases based on the structure of the tree.

Here: 4 cases derived from structure of tree

Node Methods

Fully bracketed representation of tree

```
public String toString() {
    return "(" + (left == null ? "_" : left.toString())
            + ", " + value + ", "
            + (right == null ? "_" : right.toString()) + ")";
}
```

Somewhat difficult to read, e.g.: (_, 4, ((_, 5, _), 7, (_, 8, _)))

Tree Methods

```
class Tree {
    public Node root = null;

    public String toString() {
        if (root == null) return "_";
        else return root.toString();
    }

    ...
```

Tree acts as a wrapper for Nodes (root)

Most tree methods invoke root (if not null)

Tree Methods

```
public void insert(int... values) {
    if (values != null && values.length > 0) {
        int i = 0;
        if (root == null) {
            root = new Node(values[i++]);
        }
        for (; i < values.length; i++) {
            root.insert(values[i]);
        }
    }
}
```

Insert a complete array into the tree, element by element

Tree Methods

```
    public int depth() {
        if (root == null) return 0;
        return root.depth();
    }

} // end class Tree
```

Random Permutation of Array

```
public class SearchTreeInsertion {
    private final static Random rnd = new Random(System.currentTimeMillis());

    /** Randomly shuffle the array. */
    public static void shuffle(int[] a) {
        for (int i = a.length - 1; i > 0; i--) {
            int r = rnd.nextInt(i + 1);
            int h = a[r];
            a[r] = a[i];
            a[i] = h;
        }
    }
```

nextInt(n): uniform distribution [0,n[

shuffle: each permutation equally likely

Insertion Test Function

Test function: For all expected nodes, check whether expected node is present, then check its value

```
public static void testInsert() {
    Tree t = new Tree();
    t.insert(1, 2, 3);
    boolean correct = t.root != null && t.root.value == 1 &&
        t.root.right != null && t.root.right.value == 2 &&
        t.root.right.right != null && t.root.right.right.value == 3;
    System.out.println(correct ? "test passed" : "test failed");
    ...
}
```

Aufgabe 2a (Java)

b) Bei unbalancierten Suchbäumen hängt die Tiefe des Baums von der Einfügereihenfolge der Elemente ab. In der Methode depthHistogram werden T Suchbäume mit den Elementen 1, 2, ..., N erzeugt. Die Elemente werden in einer zufälligen Reihenfolge in den Baum eingefügt. Für jeden Suchbaum wird die Tiefe bestimmt und im Array depths gespeichert. Außerdem wird die minimale und maximale gemessene Tiefe bestimmt (minDepth, maxDepth). Geben Sie ein Histogramm aus, dass für jede gemessene Tiefe die Anzahl der Bäume angibt, die diese Tiefe haben. Das genaue Format ist im Quelltext zu finden.

Depth Histogram

```
public static void depthHistogram() {
    // create an array of size N with values 1, 2, ..., N
    int N = 10000;
    int[] a = new int[N];
    for (int i = 0; i < N; i++) {
        a[i] = i + 1;
    }
```

Depth Histogram

```
int T = 250;                              // compute T search trees
int[] depths = new int[T];                // each initialized with a random
int minDepth = N + 1;                     // permutation of elements 1..N
int maxDepth = 0;
for (int i = 0; i < T; i++) {
    shuffle(a);
    Tree t = new Tree();
    t.insert(a);
    depths[i] = t.depth();
    maxDepth = Math.max(depths[i], maxDepth);
    minDepth = Math.min(depths[i], minDepth);
}
System.out.printf("min depth = %d, max depth = %d\n", minDepth, maxDepth);
```

Depth Histogram

```
// generate a histogram of this kind:
// 11: **** 4
// 12: ** 2
// 13: ******* 7
// 14: ***** 5
// 15: ** 2
//
// Generate a histogram that shows how often each tree depth occurred. In the
// example above, depth 11 occurred 4 times. The minimum depth was 11 the
// maximum depth was 15.
```

→ For each depth from minDepth..maxDepth, count how often it occurred

Depth Histogram

→ For each depth from minDepth..maxDepth, count how often it occurred

```
int[] counts = new int[maxDepth - minDepth + 1];
for (int i = 0; i < T; i++) {
    counts[depths[i] - minDepth]++;                index shifting
}
```

→ output the result as a histogram

Depth Histogram

→ Output the result as a histogram

```
for (int i = 0; i < counts.length; i++) {
    System.out.printf("%d: ", i + minDepth);
    for (int j = 0; j < counts[i]; j++) {
        System.out.print("*");
    }
    System.out.printf(" %d\n", counts[i]);
}
```

11: **** 4

index shifting

Example Histogram

for

N = 10000 (# elements in each tree)

T = 250 (# trees)

```
min depth = 26, max depth = 39
26: * 1
27:  0
28: **************** 16
29: ************************************* 37
30: ***************************************** 41
31: ********************************************************************** 70
32: **************************************** 40
33: **************** 16
34: ********* 9
35: ************ 12
36: *** 3
37: *** 3
38: * 1
39: * 1
```