

# Data Structures and Algorithms

## Exercise 8

Markus Innerebner, Romans Kasperovics  
dsa(a]inf.unibz.it

Free University of Bozen - Bolzano, Italy

Thursday, April 16th, 2009

## Assignment 08

### Task1 - Benchmark table

Use  $m = 65536$  and the load factor from the table. The number of probes will increase as the hash table is filled, so you must wait few minutes for the last step to finish. Show the experimental benchmark results in a table of the following form.

	[sec] Chaining		[sec] Linear Probing		[sec] BST	
load factor	insert	search	insert	search	insert	search
1 %	...	...	...	...	...	...
10 %	...	...	...	...	...	...
20 %	...	...	...	...	...	...
30 %	...	...	...	...	...	...
40 %	...	...	...	...	...	...
50 %	...	...	...	...	...	...
60 %	...	...	...	...	...	...
70 %	...	...	...	...	...	...
80 %	...	...	...	...	...	...
90 %	...	...	...	...	...	...

## Assignment 08

### Task 1

Implement benchmark code in C as follows to compare the performance of hash table with chaining vs. hash table with linear probing vs. a binary search tree (BST). The size of both hash tables is  $m$ . The code fills all three data structures in  $s$  steps with unique integers according to the given load factor. For instance, in the first step we fill the data structures with  $0.01 \times m$  integers. After each step the code searches for  $loadfactor \times m$  integers we know are present already (successful searches). Measure inside your code the wall clock time for the insertion and search after each step. For all data structures the code for insertion and search is attached at the end of the assignment.

## Assignment 08

### Task1 - Hash function

Use the following hash function  $h(k) = \lfloor m(kA \bmod 1) \rfloor$  where  $A = (\sqrt{5} - 1)/2$  and  $kA \bmod 1$  returns the fractional part of  $kA$ .

## Hints

### Chaining - Insertion and Search

```
void insertCH(EL* table[], int key) {
    int i = hash(key);
    EL* el = malloc(sizeof(EL));
    el->value = key; el->next = table[i];
    table[i] = el;
}

bool searchCH(EL* table[], int key, int *probes) {
    int probe = hash(key);
    EL* el = (EL*)table[probe];
    for ((*probes)++; el!=NULL; el = el->next) {
        if(el->value==key) return TRUE;
        (*probes)++;
    }
    return FALSE;
}
```

## Hints

### Linear Probing - Insertion and Search

```
bool insertLP(int* table, int size, int key) {
    int probe = hash(key); int trials = 0;
    while(table[probe]!=EMPTY && trials<size) {
        probe = (probe+1) % size; trials++;
    }
    if(trials<size) {
        table[probe] = key; return TRUE;
    } return FALSE; //table is full entirely
}

bool searchLP(int* table[], int size, int key, int* probes) {
    int probe = hash(key);
    int i;
    for (i=0; i<size; i++) {
        (*probes)++;
        if (((int)table[(probe+i)%size]) == key) return TRUE;
        if (((int)table[(probe+i)%size]) == EMPTY) return FALSE;
    } return FALSE;
}
```

## Hints

### BST - Insertion and Search

```
NODE* mkNodeBT(int val) {
    NODE* p = (NODE*)malloc(sizeof(NODE));
    p->left = NULL; p->right = NULL; p->val = val;
    return p;
}

NODE* insertBT(NODE* p, NODE* x) {
    // precondition: x not in tree p; x points to 1 node tree
    if (p == NULL) return x;
    else if (p->val > x->val) p->left = insertBT(p->left, x);
    else p->right = insertBT(p->right, x);
    return p;
}

NODE* searchBT(NODE* p, int val) {
    if (p == NULL || p->val == val) return p;
    else if (p->val > val) return searchBT(p->left, val);
    else return searchBT(p->right, val);
}
```

## Hints

### Time measurement

```
#include <time.h>
#include <stdio.h>
int main() {
    clock_t start, stop;
    start = clock();
    // put code to be measured
    stop = clock();
    double diff = (double) (stop-start)/CLOCKS_PER_SEC;
    printf("%f", diff);
    return 0;
}
```