

Data Structures and Algorithms

Exercise 8

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

Free University of Bozen - Bolzano, Italy

Thursday, April 23rd, 2009

Solutions 08

Task 1 - Data preparation

```
int* generate(bool random, int size){
    int* keys = (int) malloc(sizeof(int)*size); int i;
    if(random){
        for(i=0; i<size; i++) keys[i]=EMPTY;
        printf("# Generating %d random unique numbers\n", size);
        for (i=0; i<size; i++) {
            int probe = fabs(rand()% size); // generating position
            int trials = 0;
            while(keys[probe]!=EMPTY && trials<size) {
                probe = (probe+1) % size; trials++;
            }
            keys[probe]=i;
        }
    } else {
        printf("# Generating %d ascendent unique numbers\n", size);
        for(i=0; i<size; i++) keys[i]=i;
    }
    return keys;
}
```

Solutions 08

Task 1 - Data structures initialization - Hash function

```
void init(EL* tableCH[], int* tableLP, NODE** root, int size){  
    int i;  
    for (i=0; i<size; i++) {  
        tableCH[i]=NULL;  
        tableLP[i]=EMPTY;  
    }  
    *root = NULL;  
}  
  
int hash(int key, int size) {  
    double a = ((sqrt(5)-1)/2) * key;  
    double r = modf(a, &a);  
    return size * r;  
}
```

Assignment 08

Task1 - Benchmark code - Chaining

```
for(i=0; i<LF_SIZE; i++) { ///
    int lfsize = (int) SIZE * lfs[i];
...

start = clock(); // Benchmark CH
for (j = idx; j < lfsize; j++)
    insertCH(tableCH, SIZE, keys[j]);
stop = clock(); tInsertCH = (double)(stop-start)/CLOCKS_PER_SEC;
start = clock(); // successful search
for (j = 0; j < TEST_SIZE; j++)
    searchCH(tableCH, SIZE, keys[rand()%lfsize], &sprobesCH);
stop = clock(); tSearchCH_S = (double)(stop-start)/CLOCKS_PER_SEC;
start = clock(); // unsuccessful search
for (j = 0; j < TEST_SIZE; j++)
    searchCH(tableCH, SIZE, keys[randomLargerThan(lfsize,SIZE)], &sprobesCH);
stop = clock(); tSearchCH_U = (double)(stop-start)/CLOCKS_PER_SEC;
...

```

Assignment 08

Task1 - Benchmark code - Linear Probing

```
for(i=0; i<LF_SIZE; i++) { //f
    int lfsize = (int) SIZE * lfs[j];
    ..

    start = clock(); // Benchmark LP
    for (j = idx; j < lfsize; j++)
        insertLP(tableLP, SIZE, keys[j]);
    stop = clock(); tInsertLP = (double)(stop-start)/CLOCKS_PER_SEC;
    start = clock(); // successful search
    for (j = 0; j < TEST_SIZE; j++)
        searchLP(tableLP, SIZE, keys[rand()%lfsize], &sprobesLP);
    stop = clock(); tSearchLP_S = (double)(stop-start)/CLOCKS_PER_SEC;
    start = clock(); // unsuccessful search
    for (j = 0; j < TEST_SIZE; j++)
        searchLP(tableLP, SIZE, keys[randomLargerThan(lfsize,SIZE)], &sprobesLP);
    stop = clock(); tSearchLP_U = (double)(stop-start)/CLOCKS_PER_SEC;
```

Assignment 08

Task1 - Benchmark code - BST

```
for(i=0; i<LF_SIZE; i++) { //f
  int lfsize = (int) SIZE * lfs[j];
  ..

  start = clock(); // Benchmark BST
  for (j = idx; j < lfsize; j++)
    root = insertBT(root, (NODE*)mkNodeBT(keys[j]));
  stop = clock(); tInsertBT = (double)(stop-start)/CLOCKS_PER_SEC;
  start = clock(); // successful search
  for (j = 0; j < TEST_SIZE; j++)
    searchBT(root, keys[rand()%lfsize]);
  stop = clock(); tSearchBT_S = (double)(stop-start)/CLOCKS_PER_SEC;
  start = clock(); // unsuccessful search
  for (j = 0; j < TEST_SIZE; j++)
    searchBT(root, keys[randomLargerThan(lfsize,SIZE)]);
  stop = clock(); tSearchBT_U = (double)(stop-start)/CLOCKS_PER_SEC;
  ..
```

Assignment 08

Task1 - Benchmark table

random data generation

m=1000000

	[sec] Chaining			[sec] Linear Probing			[sec] BST		
<i>LF</i>	<i>insert</i>	<i>search</i>		<i>insert</i>	<i>search</i>		<i>insert</i>	<i>search</i>	
		<i>success</i>	<i>unsuccess</i>		<i>success</i>	<i>unsuccess</i>		<i>success</i>	<i>unsuccess</i>
1 %	0.01	0.05	0.09	0.00	0.05	0.09	0.01	0.10	0.16
10 %	0.01	0.09	0.10	0.01	0.07	0.10	0.05	0.17	0.23
20 %	0.02	0.11	0.10	0.01	0.08	0.10	0.08	0.26	0.31
30 %	0.03	0.13	0.11	0.02	0.08	0.11	0.11	0.28	0.35
40 %	0.03	0.12	0.12	0.02	0.08	0.11	0.11	0.32	0.38
50 %	0.02	0.12	0.13	0.02	0.08	0.12	0.12	0.36	0.40
60 %	0.02	0.12	0.15	0.02	0.08	0.12	0.14	0.36	0.45
70 %	0.01	0.12	0.18	0.02	0.09	0.13	0.14	0.38	0.47
80 %	0.02	0.14	0.18	0.02	0.09	0.16	0.14	0.39	0.49
90 %	0.02	0.14	0.25	0.02	0.10	0.22	0.15	0.39	0.57
95 %	0.01	0.13	0.39	0.01	0.10	0.37	0.07	0.41	0.68
99 %	0.01	0.13	1.40	0.01	0.08	1.35	0.06	0.40	1.61

Hashing: constant access time

BST: logarithmic access time (if uniform data distribution)

Assignment 08

Task1 - Benchmark table

Ascendent data generation

$m=100000$

<i>LF</i>	[sec] Chaining			[sec] Linear Probing			[sec] BST		
	<i>insert</i>	search		<i>insert</i>	search		<i>insert</i>	search	
		<i>success</i>	<i>unsuccess</i>		<i>success</i>	<i>unsuccess</i>		<i>success</i>	<i>unsuccess</i>
1 %	0.00	0.00	0.01	0.00	0.01	0.00	0.02	0.20	0.36
10 %	0.00	0.00	0.01	0.00	0.01	0.00	0.65	2.01	4.04
20 %	0.00	0.01	0.00	0.00	0.01	0.00	1.98	4.06	8.11
30 %	0.00	0.00	0.01	0.00	0.00	0.01	3.32	6.03	12.25
40 %	0.00	0.00	0.01	0.00	0.01	0.00	4.93	8.18	17.27
50 %	0.00	0.01	0.00	0.01	0.00	0.01	7.12	10.60	22.10
60 %	0.01	0.00	0.01	0.01	0.00	0.01	9.26	13.09	27.95
70 %	0.00	0.01	0.01	0.00	0.01	0.00	12.24	15.79	34.21
80 %	0.01	0.01	0.02	0.00	0.00	0.02	15.83	18.32	42.28
90 %	0.00	0.00	0.03	0.00	0.00	0.02	20.21	21.42	50.61
95 %	0.00	0.01	0.03	0.00	0.01	0.03	11.94	24.03	56.97
99 %	0.00	0.01	0.14	0.00	0.00	0.15	10.25	25.13	62.39

Hashing: constant access time

BST: linear access time (if degenerated tree)