Assignment Brief

Data Structures and Algorithms

COS5021-B

Coursework-1 (50%)

The University of Bradford
Faculty of Engineering & Digital Technologies
School of Computer Science, AI, and Electronics

Coursework-1 (50%)

Module Title: Data Structures and Algorithms

Module Code: COS5021-B
Module Credit: 20
Level: L5
Academic Year: 2024/25

Date due: 26 Nov 2024, 15:00

Length limit: There is no limit whereas each question should be explained for each steps performed.

Learning Outcomes:

This assessment will evaluate follow learning outcomes of the module:

LO1: Recognise common data structures and fundamental algorithms and be familiar with the associated terminology.

LO2: Define Abstract Data Types in terms of their data structures.

Assessment Methods:
Coursework - This consists of six (06) questions including their sub-parts. You need to answer all the questions according to the question asked**. Note:** Understanding the questions is part of the assessment and no AI tools would be used for this assessment.

Please take note of the following regulations/advice where appropriate:
- Check the assignment marking criteria when doing your assessment.
- Go through your lecture/lab materials on the canvas for more understanding.

Guidelines:

Each student is given a (different) list of numbers to be used, in sequence, as keys for entries into a heap, into a hash table and into an AVL tree.

Your assignment is to show the trace of the operations of insertion, re-structuring, etc. involved for your list of keys when they are entered into a heap, a hash table and an AVL tree.

Form of Submission

Your submission should be.

- in a plain text file
- uploaded in Canvas module assessment section.
- The content of your submitted file should be as follows.

1. Have your UoB in the first line, e.g.

   ```
   DSA Coursework          16012345
   ```

2. The word 'data' and your assigned list of keys, e.g.

   ```
   keys  25, 10, 15, ...
   ```

   Find your keys against your name and UB number in Data file.

I recommend that you copy and paste the whole line on to your submission document. When you paste the line, it will be quite noticeable if it shows someone else's UB number. You can then safely delete your UB number from the copy, knowing you have the right data.

1. Binary Heap

The task here is to show a trace of the operations needed to insert objects with your (list of) keys, one by one, into an initially empty miniHeap with restoration of heap order (if necessary) with each insertion.

Your submission should have the section heading 'Heap trace' followed by the coded trace of operations:

- `Hx` to create a *hole* at location `x` in the heap;
- `X` to move the *hole* up the heap by swapping the *hole* with its parent;
- `Ixx` to insert key `xx` into the *hole*;

with the coded operations in sequence on successive lines, e.g.

```
Heap trace
H1
I25
H2
X
I10
H3
I15
...
```

2. Heap Build

For this part, your keys are loaded into the heap without application of heap order. Show a trace of the operations in the heapBuild process, to apply heap order to the (loaded) heap.

Your submission should have the section heading 'Heap Build trace' followed by the coded trace of operations:

- `Xxx` to swap the object having key `xx` with its parent;

with the coded operations in sequence on successive lines, e.g.

```
Heap Build trace
X15
...
```

3. Heap Sort

This part is for heap maintenance during the output phase of heap sort, i.e. you begin with the heap resulting from the previous section (2. Heap Build). Show a trace of the operations to output each object (from the root) and then to restore heap order after each output.

Your submission should have the section heading 'Heap Sort trace' followed by the coded trace of operations:

- `Mxx` to remove the object with key `xx` from the root (and so create a *hole* at the root);
- `L` to move the *hole* down the heap by swapping the *hole* with its left child;
- `R` to move the *hole* down the heap by swapping the *hole* with its right child;
- `Xxx` to move object having key `xx` into the *hole* (and delete the node that previously contained key `xx`);

with the coded operations in sequence on successive lines, e.g.

```
Heap trace
M10
L
L
X57
M12
R
X39
M15
L
L
X84
...
```

4. AVL Tree

The task here is to show a trace of the operations needed to insert objects with your (list of) keys, one by one, into an initially empty AVL tree with restoration of AVL balance (if necessary) after each insertion.

Your submission should have the section heading 'AVL trace' followed by the coded trace of operations:

- `Ixx` to insert key `xx` at the root of the previously empty AVL tree;
- `IxxLyy` to insert key `xx` as the left child of the node containing key `yy`;
- `IxxRyy` to insert key `xx` as the right child of the node containing key `yy`;
- `Rxx` to rotate the node containing key `xx` with that of its parent (in order to restore AVL balance);

with the coded operations in sequence on successive lines, e.g.

```
AVL trace
I25
I10L25
I15R10
R15
R15
...
```

5. Hash Table (1)

The task here is to show a trace of the operations needed to insert objects with your (list of) keys, one by one, into an initially empty 11-bucket hash table with

- (primary) hash function `h1(x) = x mod 11` (see table at the end of this page)
- and using linear probing for collision resolution.

Your submission should have the section heading 'Hash Table trace 1' followed by the coded trace of operations:

- `Pn` to probe bucket `n` (to see if it already contains an entry);
- `Ixx@n` to insert key `xx` into bucket `n`;

with the coded operations in sequence on successive lines, e.g.

```
Hash Table trace
P3
I25@3
P10
I10@10
P3
P4
I14@4
...
```

6. Hash Table (2)

The task here is the same as for the previous section (5. Hash Table (1)) *except* that collision resolution is to use the secondary hash function `h2(x) = (x mod 3) + 1` (see table at the end of this page).

Your submission should have the section heading 'Hash Table trace 2' followed by the coded trace of operations (same coding as for previous section).

Hash Functions

The body of each table contains key values. The value of the hash function for a key is shown (in bold) at the head of the column in which the key appears.

primary hash function

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
11	12	13	14	15	16	17	18	19	20	21
22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	64	65
66	67	68	69	70	71	72	73	74	75	76
77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98
99										

secondary hash function

1	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98
99								

Coursework Marking Criteria

Since every student gets a different data set, it is not possible to give a sample solution to the coursework. There is a single marking criterion: correctness of the solution. However partial credit will be awarded to solutions based on what mistakes are made. It is very important that for each step of the questions in the entire coursework should be backup with its explanation. For instance, if you insert/delete a key in the tree, you should clearly explain why you have inserted/deleted the key in the respected location. May be due to a current key is less than or greater than the previous key etc. Keep in your mind a mistake at the early stage could end with wrong tree or answers and you will be deducted the marks. Double-check your current step while proceeding with the further steps.

The components of the coursework are weighted as follows:

- Binary Heap: **20%**
- Heap Build: **15%**
- Heap Sort: **15%**
- AVL Tree: **20%**
- Hash Table (1): **15%**
- Hash Table (2): **15%**

**************************************End**********************************