HEAPS, HASH

[Updated: 10:40pm 2018-02-02]
[Updated: 11:40pm 2018-01-28]

To obtain the intended learning objectives:
* Start early; Pace your work
* No group work; no team work
* Debug your code yourself

Due: your labs session the week of 4 Feb 2018

Submit via the arg "3" to the standard submit command

Python and C will be used for this lab, as per the spec.

This lab will extend various concepts presented in class;
it is thus self-contained. However, if you have a conceptual
gap that prevents you from perceiving this self-containment,
ASK ME and get your conceptual blockage fixed.

A. Introduction

We've seen how Trees can be implemented via lists (Python)
and linked structures (C). These structures have a topological
congruence to the abstract concept of a Tree. The Tree is, after
all a set of nodes that are linked to each other. Or looking at it
another way, a tree is a recursive structure: a node that has
various successors, that are themselves nodes with successors.
The former view leads to the linked structure implementation in C,
while the latter view leads to the recursive list implementation in
Python.

A linked implementation (C) does have some disadvantages, despite the
conceptual simplcity. As an exercise, print the result of applying
sizeof(), on your C linked structure, and compare it to the
sizeof() applied to the data you're storing in the tree. Observe the
large overhead of the linked structure.

The recursive tree implementation (Python) has disadvantages as well:
indexing into arbitrary levels of the tree requires recursive (or
iterative) traversal.

Do we have an alternative? This lab will explore this theme.

Let's first recall that lists (linked or otherwise) are sequential
structures. What is another sequential structure? The array.

Below, we'll see two data structures that are often implemented
via arrays.

B. Heaps

Language = C

Heaps are specializations of Binary Trees that relax the strict
ordering of Binary Search Trees: the root node is either less than or
greater than its successors. There is no relative ordering between the
successor trees.

A Min-Heap is one where the root node is less than its left or right
successors. What do you think a Max-Heap is?

Regarding implementation of Heaps. Note that with Binary Trees the
indices of the nodes at any level in the tree is clearly defined:

N=Root; L=Left; R=Right

Node	Index
N	0
N->L	1
N->R	2
N->L->L	3
N->L->R	4
N->R->L	5
N->R->R	6
N->L->L->L	7
N->L->L->R	8
N->L->R->L	9
N->L->R->R	10
N->R->L->L	11
N->R->L->R	12
N->R->R->L	13
N->R->R->R	14

and so on.

We thus have a very simple fundamental data structure
(the array) that can represent binary trees, without the
storage penalty of a linked list. (Note: you should perform
the sizeof() exercise in the intro so you can see this).

Heaps find themselves ubiquitously used, and hence this
straightforward implementation machanism is providential.

What are the limitations of using an array to represent a binary
tree?

First, one has a limit on the size of the tree: the size of the
array (if the array is statically defined, as in C) limits the number
of elements.

Second, without the NULL pointer, a minor limitation is how does
one know whether the leaf nodes are in the tree?

If one were to make a binary tree of integers, then how would one
know whether there was a legitimate data item in a node, as opposed
to it being a NULL node? If we were storing only positive integers,
we could exclude the negatives or zero and use them to identify
whether the tree node was NULL or not. But what if we were storing
general integers, for which we couldn't exclude any values?

A simple solution here is to make an array of structs:

```
typedef struct {
   int key;
   unsigned int valid;
} keyType;
```

and then define a tree with max depth 8 as:

```
keyType binary_tree[255];
```

which we would initialize by setting the valid slot of each node to 0:

```
for (i=0;i<2**8;i=i+1){
   (binary_tree[i]).valid=0;
}
```

```
When adding to the tree, we would ensure to set the valid slot to 1.

For the case of a heap, one does not need to maintain the
valid indicator as above. The insertion point of a heap is
always the last valid element in the heap. Hence, one does not
need to track or demark ALL leaf nodes, but just one.

In this case, one need only maintain an index that tracks this
insertion point.

For the following assignments, assume we're using a max heap and
define your heap via a struct that encapsulates all the requistite
info.

typedef struct {
   int *store;
   unsigned int size;
   unsigned int end;
} HeapType;

Assignment B-I:
Write the function:
   int initHeap (HeapType *pHeap,int size);
where pHeap is caller-allocated, and size indicates how large the heap should be.
What will you initialize end to be? The return value will be 0 for success and -1
otherwise.

Write the functions:
   int inorder  (HeapType *pHeap, int **output, int *o_size);
   int preorder (HeapType *pHeap, int **output, int *o_size);
   int postorder(HeapType *pHeap, int **output, int *o_size);
where:
-> ditto for pHeap and return value
-> "int **output" is the caller-unallocated array that will store the
    output data
-> "int *o_size" is the size of the output array (returned to the caller)

The function should perform the indicated traversal of the input binary
tree, and store the output in the output array; remember, output
is not caller-allocated (what does this mean? what must you do?). As well,
we don't know how many valid keys were present, so o_size must be sent
back to the caller.

Assignment B-II:
Addition of data to a BST was explored in prior labs. Here we will
write a function to add to a binary heap.

Write the function:
   int addHeap(HeapType *pHeap, int key);
that will add "key" to the binary heap stored in "input".
What happens when you add a key with a value higher than the
root node?

Write the function:
   int findHeap(HeapType *pHeap, int key):
that will look for whether there is an element key in the heap
with the return value:
   -1 for error (not okay)
   0 for okay: not found
   1 for okay: found

Write the function:
   int delHeap(HeapType *pHeap, int *key);
which will remove the max value from the max-heap pHeap, and return it to
```

```
the caller via "key".

Submit:
Put all the functions you've written in a file called heap.c and submit it.

Write test_heap.c (which will have a main function, in contrast to heap.c)
and test your code above.

C. Hash Functions

Language = Python

Write a function, f, in a file, hash.py, that will take
integer inputs and based on the input return an output
integer that is between 10 and 100.

Submit this hash.py

D. Hash Tables

Language = Python

TO BE DEFINED IN A FUTURE LAB...
```