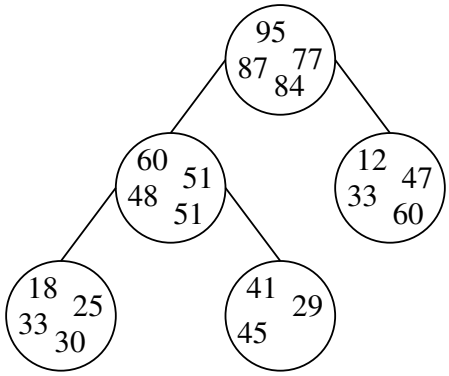
CS29003 Algorithms Laboratory

Assignment No: 8

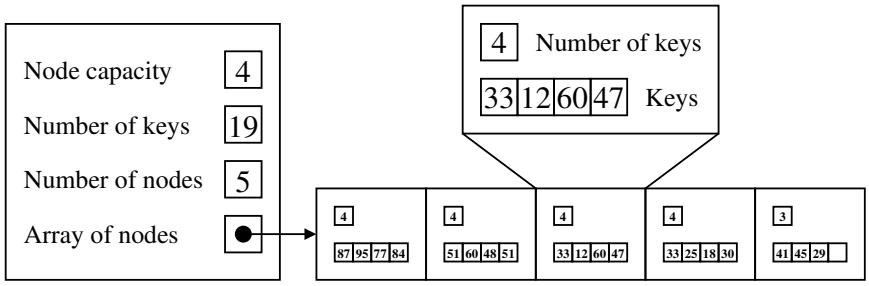
Date: 24–March–2020

Heaps and Priority Queues

This assignment deals with a max-heap (or max-priority queue) with each node storing multiple keys. Denote by p the key capacity of each node. The nodes are stored in the contiguous representation. Each node, except possibly the last, must be full (that is, must contain p keys). The keys in each node are stored in an array which is not needed to be sorted in any order (ascending or descending). The heap ordering property must hold, that is, if k is any key in any node, and k' any key in any of its child nodes, then we must have $k \geqslant k'$. Multiple keys of the same value may be present in (one or more) nodes of the heap. Let us call such a heap a *multi-heap* (although this term is used with a different meaning in other technical contexts). The following figure illustrates a multi-heap with $n = 19$ keys and with node capacity $p = 4$.



A max−heap with multiple keys per node



Representation of the multi−heap

Multi-heap data structure

The right side of the above figure shows how you represent a multi-heap. Declare suitable user-defined data types for this representation. Assume that the keys are integers (positive if you like).

Multi-heap functions

Implement the following functions for a multi-heap.

initheap(p, n_{max}) The user specifies the node capacity p and a maximum number n_{max} of keys that the multi-heap is meant to store. This function allocates appropriate memory to the array of nodes based upon n_{max}. It is your choice whether you make the indexing in this array zero-based or one-based. The function also sets the current number of keys to $n = 0$, and the current number of nodes in use to $N = 0$. At any point of time, these two counts are related as $N = \left\lceil \frac{n}{p} \right\rceil = \left\lfloor \frac{n+p-1}{p} \right\rfloor$. An empty multi-heap initialized in this manner is to be returned by this function.

insert(H, x) This function is used to insert a key x to a heap. The procedure works as follows. If the last node is full, go to the next node and insert x there, else append x to the current last node. This insertion may violate the heap-ordering property. This is repaired by moving the disturbance up toward the root. Let the current node be q. If q is the root, we are done. Otherwise, let r be the parent node of q. Compute the minimum key r_{min} in r and the maximum key q_{max} in q. If $r_{min} \geqslant q_{max}$, we are done. Otherwise, pick the largest p of the keys in r and q for relocating in r, and the remaining (smaller) keys for relocating in q. Then, proceed to r, and repeat.

findmax(H) If H is a non-empty multi-heap, then the maximum key resides in the root node. Since the key arrays in nodes are not necessarily sorted, a linear (in p) max-finding algorithm suffices.

heapify(H,i) Modify the heapify procedure for binary heaps to work for multi-heaps. Compute the minimum key q_{min} at the current node, and the maximum keys l_{max} and r_{max} at the two child nodes (if present). If necessary, reallocate the keys, and move on to the appropriate child. The *heapify* function is to be used only by *delmax*, so you may assume that there is at most one key placed out of order in the node at which you heapify.

delmax(H) Remove the largest key from the root node. If the root node is the last node, we are done. Otherwise, move any key from the last node to the root node. Call *heapify* at the root.

prnheap(H) This function prints a multi-heap in the format illustrated in the sample output.

The *main*() function

- The user enters the node capacity p and the total number n of keys to be inserted. Subsequently, the user supplies n keys. Store the keys in an array A.
- Initialize a multi-heap H, and insert in H the keys stored in A one by one.
- Print the multi-heap after all the insertions.
- Keep on calling *findmax*, storing the returned value in A (from the end to beginning), and running *delmax* on H, until H becomes empty. Print the array A (from beginning to end).

Sample output

```
10
128
407  958  777  425  129  909  423  401  962  700  410  285  910  634  203  966  131  220  927  883
457  130  521  904  829  965  927  540  375  201  927  734  211  757  211  293  666  535  594  680
287  956  918  197  590  121  215  674  241  142  557  650  172  130  607  901  996  586  494  371
739  421  157  851  178  268  144  797  703  638  477  890  646  395  139  237  469  254  811  662
348  420  313  420  450  872  374  446  458  768  769  197  241  826  948  320  995  144  169  750
682  546  641  381  894  680  518  363  887  381  925  235  753  290  556  255  162  882  654  572
702  423  722  843  302  722  163  349

+++ 128 insertions made
    [  996   995   966   965   962   958   956   948   927   927  ]
    [  927   918   909   904   894   890   887   883   872   826  ]
    [  925   910   901   882   851   843   829   797   777   753  ]
    [  674   666   662   646   594   590   477   469   458   450  ]
    [  811   769   768   757   750   734   682   680   680   641  ]
    [  739   722   722   703   702   700   654   650   572   556  ]
    [  634   607   586   557   421   268   178   157   144   638  ]
    [  285   254   237   220   211   211   203   139   131   129  ]
    [  446   420   420   407   401   395   374   348   313   293  ]
    [  287   241   241   215   197   197   169   144   142   121  ]
    [  546   535   518   425   423   410   381   363   320   381  ]
    [  375   371   290   255   235   201   172   162   130   130  ]
    [  540   521   494   457   423   302   163   349             ]

+++ 128 deletions made

+++ Input array sorted
121  129  130  130  131  139  142  144  144  157  162  163  169  172  178  197  197  201  203  211
211  215  220  235  237  241  241  254  255  268  285  287  290  293  302  313  320  348  349  363
371  374  375  381  381  395  401  407  410  420  420  421  423  423  425  446  450  457  458  469
477  494  518  521  535  540  546  556  557  572  586  590  594  607  634  638  641  646  650  654
662  666  674  680  680  682  700  702  703  722  722  734  739  750  753  757  768  769  777  797
811  826  829  843  851  872  882  883  887  890  894  901  904  909  910  918  925  927  927  927
948  956  958  962  965  966  995  996
```

Submit a single C/C++ source file. Do not use global/static variables.