CS 251
Spring 2009
Voicu Popescu
Assignment 6
Due Friday March 6, 11:59PM

Priority queues

Part 1: Implementing a min priority queue as a min heap

Implement a **Heap** class using a binary tree (and not an array) with the following functionality:

- Constructor: **Heap()**
  - Creates and initializes the heap. This function must run in O(1) time.
- Destructor: **~Heap()**
  - Deallocates any memory used and destroys the heap in O(n) time when there are n elements in the heap.
- Insert: **void insertElement(int key, int data)**
  - Puts **data** in the heap using **key** to determine its location. If there is already an entry with key **key** you will still create an entry in the heap with that **(key,data)** pair (that is, insert as you would normally). This function must run in O(log n) time.
- Minimum element: **const int minElement() const**
  - Returns the data of the element in the heap with the smallest key. This function must run in O(1) time.
- Extract minimum element: **int extractMinElement()**
  - Returns and removes the data in the heap with the smallest key. This function should ensure that the heap properties are maintained. This function must run in O(log n) time.

Additional Clarifications:

- Your Heap should have NO upper limit to the number of elements it can hold (if you had an infinite amount of memory available). That is, it should be dynamic.
- Your Heap class should be independent of the rest of the project (Part 2 below) and will be tested individually. That is, someone else should be able to use Heap in their program and have it function as a min priority queue.
- DO NOT change any of the function signatures including const modifier. Doing so will most likely lead to a compile error and a 0 for your overall program grade. Your functions must run in the stated time requirements. If they do not, then you will receive no credit for that function.
- There is no "main" for this part of the project. You are creating the Heap class which you will use in Part 2 below. The corresponding files for Part 1 are **Heap.h** and **Heap.cpp** (Skeleton code is provided). You may use the file **heaptest.cpp** to add your test cases and test Heap class. "**make heaptest**" will create the executable **heaptest**. heaptest.cpp will be replaced when grading your assignment.

Part 2: Sorting and finding the k minimum elements using a priority queue

You will create two applications in this part utilizing your Heap class from Part 1. To use Heap, you will probably want to use the number that you are storing in the heap as both the key and the data.

Application 1: heapsort

Take n integers and print them out sorted in ascending order. The running time should be O(n log n). The details are as follows:

- This program will read from a file given on the command line which contains n integers where n is greater than or equal to 1. The integers will be separated by whitespace which you will need to handle. Assume that the input will be in the proper format. An arbitrary file name can be given on the command line.
- The program will then use the Heap class and will write to a file named **heapsortoutput** the result of sorting the n numbers in ascending order. The output should be listed on one line with a single space separating numbers. No whitespace/newlines should precede the first number or come after the last number.
- Example input and output are provided in the files heapsortsampleinput and heapsortsampleoutput. Please make sure that your program can EXACTLY reproduce that output file given the input file. You may want to use the `diff' program to test this.
- A skeleton **heapsort.cpp** file is provided. "**make heapsort**" will create the executable **heapsort**.
- An example of how this application will be run is:
  - heapsort inputfile

Application 2: kmin

Take n integers and print out in sorted ascending order the k smallest among them. The running time should be O(n log n + k log n). The details are as follows:

- This program will read from a file given on the command line which contains n integers where n is greater than or equal to 1. The integers will be separated by whitespace which you will need to handle. Assume that the input will be in the proper format. An arbitrary file name can be given on the command line.
- The program will also read from the command line a number k which is the number of elements to output. k will be greater than or equal to 1 and less than or equal to n.
- The program will then use the Heap class to write to a file named **kminoutput** the result of outputting the minimum k elements in sorted ascending order. The output should be listed on one line with a single space separating numbers. No whitespace/newlines should precede the first number or come after the last number.
- Example input and output are provided in the files kminsampleinput and kminsampleoutput_k=5. Please make sure that your program can EXACTLY reproduce that output file given the input file. You may want to use the `diff' program to test this.
- A skeleton **kmin.cpp** file is provided. "**make kmin**" will create the executable **kmin**.
- An example of how this application will be run is below, where 3 is an example k
  - kmin kminsampleinput 3

Extra credit 1: **compile-time simplification of expression trees [3%]**

A binary expression tree is built from a perfectly parenthesized string like for A5 (assignment five). In addition to the possible input for A5, the expression can also contain the variables 'a', 'b', 'c', and 'd'. Simplify the tree by replacing any subtree that does not contain variables (but only numbers and can therefore be evaluated) with a leaf that stores its value. Also simplify the tree by replacing a subtree with a leaf storing 0 when the tree multiplies one of its children with the constant 0. Print your tree as in A5. The details are as follows.

- Extend the constructor of the BinaryTree class you developed for A5 to build the simplified tree. (**Hint**: Build the tree as in A5 and then prune the tree to get the simplified tree.)

- Assume that sub-expressions evaluate to less than 10 so that you don't need to change the draw method in A5.
- When **simplify-tree.cpp** (provided) is compiled, it should draw the simplified tree to tree.txt file. (Uncomment the relevant entries in the Makefile to compile it as "make simplify-tree" and produce the executable simplify-tree)
- Some examples are given below:
  - Input expression = (a + (2 * 3)) Simplified expression = (a + 6)
  - Input expression = (a + (b * 0)) Simplified expression = (a)
  - Input expression = (a + (b * 5)) Simplified expression = Input expression
  - Input expression = ((2 + 2) * (1 + 1)) Simplified expression = (8)

Extra credit 2: an efficient kmin [5%]

Implement a O(n + k log k) solution to the kmin problem (see above). It is OK to assume that n is of the form 2^i -1. Provide a high level description of your algorithm.

Hint 1: You will probably have to use bottom-up heap construction (see your text for details).
Hint 2: Think divide and conquer.
Details are as follows:

- Provide **ekmin.cpp** for this. (Uncomment the relevant entries in the Makefile to compile it as "**make ekmin**" and produce the executable **ekmin**)
- Explain your algorithm in **ekmin.txt** file.

Turn in Instructions

- Your codes MUST compile on lore.cs.purdue.edu for it to be graded.
- All of your documents (Heap.h, Heap.cpp, heapsort.cpp, kmin.cpp, BinaryTree.h (optional), BinaryTree.cpp (optional), ekmin.cpp (optional), ekmin.txt (optional)) MUST be in the directory A6.
- Read General turnin instructions parts E, F and G very carefully, on the website.
- In the directory above A6, run the command on lore.cs.purdue.edu
  turnin -c cs251 -p A6 A6
- Optionally run the following command to verify what files were submitted
  turnin -c cs251 -p A6 –v
- Warning: turnin will be automatically disabled at 03/06/2009 at 11:59 PM EST and you will be unable to turnin your assignment after this time.
- Warning: turning in multiple times is ok but only the last one will be graded as each turnin permanently overwrites the previous one.
- Warning: failure to turnin exactly according to the instructions given above will result in your A6 receiving a grade of 0.