CS 251
Fall 2007
Voicu Popescu
Assignment 6
Due Monday October 15, 11:59PM

Priority queues

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

Create a class named **Heap** in files **Heap.h** and **Heap.cc**. The Heap class will implement a min priority queue using a dynamically allocated tree structure NOT an array-based structure. Heap will provide 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. This function must run 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 when there are n elements in the heap. In general, DO NOT assume that key == data although this will probably be true in Part 2 below.
- 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 when there are n elements in the heap.

Additional clarifications:

- You will receive NO CREDIT if you implement an array-based heap.
- 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. 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.

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

This application code will go in the file **heapsort.cc** and the corresponding executable file will be **heapsort**. The basic goal of this application is to take n integers and print them out sorted in ascending order. See below for details.

- 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.
- This program must run in time O(n log n).
- 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.
- An example of how this application will be run is:
    - heapsort inputfile

Application 2: kmin

This application code will go in the file **kmin.cc** and the corresponding executable file will be **kmin.** The basic goal of this application is to take n integers and print out in sorted ascending order the k smallest among them. See below for details.

- 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.
- This program must run in time O(n log n + k log n).
- Example input and output are provided in the files kminsampleinput and kminsampleoutput_k=3. 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.
- An example of how this application will be run is below, where 3 is an example k
    - kmin kminsampleinput 3

You are provided skeleton code and a Makefile. You can compile heapsort with the command “make heapsort”, kmin with the command “make kmin”, or both with the command “make”.

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

Create a file named **simplify-tree.cc** and corresponding executable file **simplify-tree.** Edit the makefile so that it includes a rule so that the program simply-tree can be created with the command “make simplyify-tree”. Read the input string from standard input and build a binary tree 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 to the file **simplified-tree-output**. You should include your binary tree code which will be in files **BinaryTree.h** and **BinaryTree.cc.** You do not need to submit any additional files for this to be graded.

Extra credit 2: an efficient kmin [5%]

Create a file named **kminec.cc** and the corresponding executable file **kminec.** Edit the makefile so that it includes a rule so that the program kminec can be created with the command “make kminec”. This program will be **identical** to kmin in application 2 EXCEPT that this version MUST run in time:

- O(n + k log k)

for a file containing n numbers and a given k on the command line. It is OK to assume that n is of the form 2^i -1. You must also explain your algorithm in a high level of detail in the file **kminec.txt.** No extra credit will be given for programs not running in the desired time bound. You do not need to submit any additional files for this to be graded.

Warning: This extra credit should only be attempted after you complete the project. Attempting to “pass off” your application 2 kmin as a solution for this will result in a 0 project grade.
Hint 1: You will probably have to use bottom-up heap construction (see your text for details).
Hint 2: Think divide and conquer.
Hint 3: Good luck!

- Turn in instructions:
    - Assignment specific:
        - Turn in files: Heap.h, Heap.cc, kmin.cc, heapsort.cc, Makefile, kminec.cc(if applicable), kminec.txt(if applicable), BinaryTree.h(if applicable), BinaryTree.cc(if applicable), simplify-tree.cc(if applicable)
        - **Your code is REQUIRED to compile with /opt/csw/gcc3/bin/g++ (gcc version 3.4.5) on the borg machines. Please update your Makefile or use the one that is provided with the assignment. If your code doesn't compile with this compiler, then even if it compiles with another, your code will still be considered noncompiling.**
        - Directory structure: all files in the directory lab6
        - turnin command executed in the directory containing your lab6 directory:
            - turnin -c cs251 -p lab6 lab6
    - General:

      See instruction at http://www.cs.purdue.edu/cgvlab/courses/251/As/turnin.html
      Make sure to check it EVERY TIME before submitting a new assignment. It is updated periodically.