EC 504
Spring, 2021
HW 2 Software

Due Friday, 2/26/21, 8 p

Note: This must be turned in by creating a HW2 folder in your directory in /projectnb/ec504 directory on the SCC cluster (scc1.bu.edu). Please turn in the output file and the source files for your code. Repeat what you did in HW0 and HW1.

1. (30 pts) In this exercise, you are to implement an unusual min-heap binary priority queue. As mentioned in class, the standard class implementations of heaps often fall short of what is needed for algorithms, and are slow. The main program, `main.cpp` , creates an array of records called heapItems, with keys. Each record also holds an index field, `position` , that points to its current in the heap array.

   The heap elements will be pointers to these records. This means that there will be a lot less copying of data when heap elements are swapped. It also allows one to update the position field from within the heap class using the pointer. What it also means is that you can't simply go copy a heap implementation for integers on the web and expect it to work. You don't want to organize the heap by the value of the pointers; you want to organize it by the key values of the records. From the heap class, you can access this as `array[k]->key` , using the pointer stored in the heap array to get the key value. Similarly, you access the position information as `array[k]->position` . As you do heap operations that move pointers in the heap, you need to make sure you update the position of records in the heap. When a record is removed from the heap, set its position to -1.

   Included in the files is an include file, `myHeap.h` , and a makefile. You need to fill out all the missing functions in `myHeap.h` . To make an executable, type `make heapmoves` . Run the executable file, `heapmoves` , using the command `./heapmoves >> heap.out`

   Turn in files `myHeap.h, heap.out` in a HW2 folder in your directory in /projectnb/ec504 dirrectory. Do not include any other files.

   Note: We will take your `myHeap.h` file and compile it against a much more complex main.cpp that uses larger heaps and many other operations for validation. We will test that heap order is maintained throughout, and we will check that every record knows its correct position in the heap.

   the heap property by the value of the pointers:

   What we will do is insert all of the read records into an array of pointers to records. Then, we will build a max-heap from those pointers, using the numeric value of the zip code. The main program will ask some simple questions, such as find the k-th largest item in the zip code list for several values of k (ordered, so you can be removing elements from the heap). It will do so by calling the remove-max routine a fixed number of times, removing elements one at a time.

   A main program is provided, with many functions missing. You are to upload the main program with all the functions completed. Do not make any changes to the input and output reading formats in main() so the Automatic Grader does not get confused running it against a set of input filles. The input files in the web site are small examples for debugging. Your program will be evaluated against larger files in the Automatic Grader.

2. (20 pts) The purpose of this exercise is to implement the Knuth-Morris-Platt (KMP) string matching algorithm. Pseudocode for this algorithm is provided in our class slides. You will get an input of two strings: a pattern string and a text string. You are to find the locations of all positions where the pattern matches the string. Enclosed is a trivial main program, `KMP.cpp` , and a header file `KMP.h` that is missing two functions for you to complete: `computePiArray` needs to compute the prefix function $\pi(\cdot)$ for the pattern string, and `KMPSearch.cpp` , which uses this pattern to find the matches using the KMP algorithm.

   To make, call `make KMP` . To run, call `./KMP` . Use the provided main program to debug. Feel free to edit the pattern and text to make sure your program works. Turn in your completed `KMP.h` file in the HW2 folder submission, along with the output of the previous problem. We will use your include file with our main program with a much longer string and harder pattern.