CS 251, Fall 2007
Voicu Popescu
Assignment 7
Due Monday October 22, 11:59PM

Hash Tables

1. Implement a C++ hash table class **HashTable** in files HashTable.cc and HashTable.h
    a. The items stored are integers which serve both as keys and as elements.
    b. The hash table uses double hashing
        i. The primary hashing function is $h(k) = k \bmod N$, where k is the key, and N is a prime positive integer provided as input. Thus your table will have room for N entries.
        ii. Collisions are resolved using a secondary hashing function $h_2(k) = (h(k) + j * (q - k \bmod q)) \bmod N$, where $j = 0, 1, \ldots, N-1$, and q is a prime positive integer provided as input.
    c. The constructor **HashTable**(int *N*, int *q*, char ***filename**) builds the hash table from keys stored in the file **filename**. The parameters N and q were defined above. The file is a text file with $n+1$ space and/or new line delimited positive integers. The first integer is n, which gives the number of keys. The following n integers are the n keys.
    d. The void **Insert**(int *k*) method inserts the key k using the double hashing described above if the table is not full, and does not do anything if the table is full.
    e. The int **Find**(int *k*) method returns the index of a table entry that stores key k, if one exists, and -1 otherwise.
    f. The void **Remove**(int *k*) method makes available a table entry that contains the key k and does not do anything if the table does not contain key k.
    g. The void **Print**(char ***filename**) method writes in a file called **filename** all table entries in ascending order of the table slot number, one per line. An entry that is available is printed as **index** "-1", where index is the entry's index in the table. An entry that stores a key is printed as **index key**. See the sample input/output files for exact formatting examples.
    h. The destructor **~HashTable()** deallocates any used memory.
    i. Additional instructions
        i. Example input and output are provided in the files hashtablesampleinput and hashtablesampleoutput_N=5_q=7. The example output is for $N = 5$ and $q = 7$. 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.
        ii. 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.
        iii. There is no "main" for this part of the project. You are creating the HashTable class which then will be tested for correctness.
        iv. Be sure to test your program thoroughly.
        v. A Makefile is not required.
        vi. Do not partition your code into files we haven't asked for.

2. Number of probes.
    a. For $N = 7{,}919$ and $n = 3{,}000$, what is the expected number of probes per insert, assuming the n numbers are randomly selected? Write your answer in a plain text file named "q2a.txt". When writing math, make it legible. For example, n(n+1)/2 and O(n^2) are ok. O(n2) isn't the same as O(n^2).

b. Measure the maximum and average number of probes per insert for pseudo-randomly generated numbers and for q = 1,493. That is, run a test and report your results. Write your answer in a plain text file named “q2b.txt”.

3. Extra-credit [4%] Given a set of n unit squares defined on a regular 2D grid, eliminate all internal edges (see Figure 1) in worst-case linear time (in n) using a hash table. The set of unit squares is dense, meaning that A is $O(n)$, where A is the area of the axis aligned bounding box of the squares. In a file ec.cc, create a program with a main function with the following functionality.
    a. A square is defined by the bottom left vertex (x_0, y_0); the other three vertices are (x_0+1, y_0), (x_0, y_0+1), and (x_0+1, y_0+1). All will be nonnegative numbers.
    b. An edge is a unit segment $[(x, y), (x, y+1)]$ or $[(x, y), (x+1, y)]$ that belongs to at least one unit square in the given set.
    c. An internal edge is an edge shared by 2 squares.
    d. The program should take an input file name as the first parameter and an output file name as the second parameter on the command line. The input file is in text format and contains 1+2*n positive integers, delimited by space and or new lines. The first number is the number of unit squares, the remaining numbers specify the bottom left vertices for the n unit squares. The output format is similar to the output format for the Print function above. Output the segments in any order. See the sample input and output files ecsampleinput and ecsampleoutput for details.
    e. A Makefile is not required.

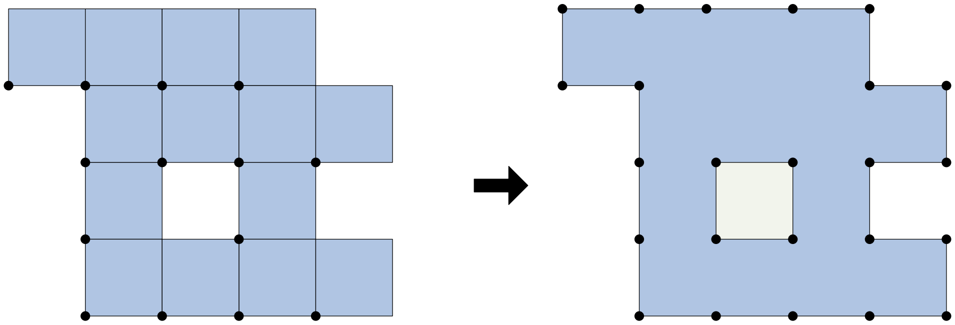

Figure 1. The set of 14 unit squares (*left*) is converted to the contour segments (*right*).

4. Turn in instructions:
    a. Assignment specific:
        i. Turn in files: HashTable.cc, HashTable.h, q2a.txt, q2b.txt, ec.cc (if applicable).
        ii. Your code is REQUIRED to compile with /opt/csw/gcc3/bin/g++ (gcc version 3.4.5) on the borg machines. If your code doesn't compile with this compiler, then even if it compiles with another, your code will still be considered noncompiling.
        iii. Remember that assignment-specific turnin instructions override general instructions.
        iv. Directory structure: all files in the directory lab7.
        v. turnin command executed in the directory containing your lab7 directory:
            - turnin -c cs251 -p lab7 lab7.
    b. 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.