CS 130A - Programming Assignment 1

Due on May 1

1 Perfect Hashing

In the first exercise you will implement perfect hashing as described in the lecture notes. Perfect hashing uses the notion of universal hashing.

Universal Hashing: In universal hashing, if we have n binary keys with size u bits each, we want to create a hash table H with $m = 2^b$ rows/bins. The hashing function will be a function h(x) = dec(Hx) mod m, where x is the key vector of binaries of size u-by-1, H is a b-by-u matrix, dec() represents conversion to decimal, and mod m gives the remainder after division with m.

Perfect Hashing - Level 1: If we perform the aforementioned process for m = n, then the data will be hashed in an imperfect way, in that there is some probability that more than one keys will be hashed to the same value, which naturally means that some bins will be empty. Usually, we perform universal hashing repeatedly until the collisions are not many.

Perfect Hashing - Level 2: Then, we visit every non-empty bin and we hash anew the data using universal hashing again, with $n = n_j$ input keys, being the keys that were hashed to this bin in the first level, and $m = n_j^2$ new bins (or the next power of 2, see second remark). The second-level hashing has to be applied repeatedly until there are no collisions happening in the j-th hash table. The second-level hashing in a bin where only one element was hashed in the first level obviously yields no collisions. The second-level hashing in a bin where more than one elements were hashed should be repeated only few times, since the probability of collision is less than 1/2. For more details, see section 5.7.1 of the textbook, or section 10.5 here `https://www.cs.cmu.edu/~avrim/451f11/lectures/lect1004.pdf`

Your task: You will be given a sequence of numbers and you will have to implement perfect hashing, following the steps below.

1. First, you will count the data items from the text file you will be provided with.
2. Then, you will create a hash table that creates a random matrix with the afore-described specifications, hash the data, and count the collisions.
3. Then, visit every bin.
   - If no element has been hashed to this bin, there is no need to hash again since there are no keys to be hashed.

- If only one element has been hashed to this bin, second-level hashing is trivial and the key is hashed to the one and only row of the new hash table, so again no need to hash the bin.
- If more than one elements are hashed to this bin, then create a hash table with the specifications described above, hash the data, and count the collisions. Do this repeatedly until NO collisions happen.

Do this repeatedly until the total space usage is less than 4n.

Remarks:

- The perfect hashing method is not flexible, in the sense that, once all hash matrices have been determined, no new key can be hashed with the same hash matrices.
- The height of the hash matrices is the log_2 of the amount of keys that we want to hash. If the amount of keys is not a power of 2, then the height of the matrix is the next integer. For instance, if in level 1 $n_j = 2$ values have been hashed to a bin, then the corresponding hash table should be of size $n_j^2 = 4$, which is a power of 2, and therefore the height of the corresponding hash matrix should be 2. Howerver, if $n_j = 3$, then $n_j^2 = 9$, which is not a power of 2. Hence, the height of the hash matrix should be equal to 4, but hashing function will be *key mod 9*. This will result in a hash table of size n_j^2.
- Your code should also support the *search* operation. Given a key, the code should return the bin in the first level and, if needed, the bin in the second level.

Input: The input of the program is a text file where every line is a binary key. See input files `hw1_q1_in1.txt` and `hw1_q1_in2.txt`. Both input files contain an amount of keys that is a power of 2, so no need to worry about the second remark, at least for the first level.

Output: The output of the program is the hash matrix of the first-level hashing as well as the hash matrices of the second-level hashing, that your code generated. The hash matrices should be separated by an empty line. For the bins that did not require further hashing (i.e. the bins that no key or only one key was hashed) write the string NULL. `hw1_q1_out0.txt` is an example of how the output file should look like. These matrices should be followed by the total space used (which is the sum of the sizes of all hash tables, in entries, not in bytes) and by the number of accesses for the search operation of each element, in the order they are given in the text file. Clearly, the number of accesses should be either 1 (if no other elements were mapped to the same bin during level 1 hashing) or 2 (if level 2 hashing was required for this bin).

What to turn in: Your commented code, a short report explaining what you did and how one can run your code, as well as the output files that your code generates when the two input files are given, with the format `hw1_q1_out1.txt` and `hw1_q1_out2.txt`.

Grading: To receive full grade, the matrices in the output files should be such that hashing is indeed perfect, total space usage is less than 4n, and the reported number of accesses for each element is correct, given the hash matrices that you provided. However, even if your code does not run perfectly (or at all), don't worry too much, you will receive partial credit.

2 Hash Table and Max Heap

In the second exercise, you will implement a multi-level caching scheme. Such schemes are used in the cache hierarchy as well as in a Translation Lookaside Buffer (TLB).

Implement the first level of cache as a Hash Table (HT) with linear probing and the second level as a Max Heap (MH). Each data element has three fields: 1. address of the element (key) 2. value of the element 3. priority of the element (representing its importance as in its frequency of recent accesses for a TLB).

When the user makes a query for an element, the HT is first queried using the key. If there is hit, the user can access the value present in the corresponding entry. If there is a miss, then the user needs to look for the the element in the MH.
The data structure should support the following operations:

1. Load_MH : Load the values from a specified input file into the MH. The priorities are set to 0. HT is initially empty.
2. Load_HT : Move the top k (size of HT) priority elements to HT. Some elements from the HT may need to be moved to the MH. If the priorities are same, then the element with higher key value will be present in HT.
3. Access(key) : Find the specified element, increase its priority by 1 and print whether it is present in HT/MH along with value of the element.
   - Print format : (0/1/-1) <value of the element>
     a) 0 : if present in HT
     b) 1: if present in MQ
     c) -1 : if element is not present
4. Display : Returns all the elements present in the HT/MH.

Your Task

1. You should first load, the data that is given to you into Max Heap.
2. Then, you move the top k(size of HT) priority elements to HT.
3. Then, whenever query has been made access the specified element by above aforementioned process.

Remarks: First step to perform after reading all the inputs is Load_HT.
Input hw1_q2_in.txt : This file contains the elements that need to be loaded followed by queries

(a) First line : Number of test cases. (nT)

(b) Second line : HashTable size of the test case. (sHT)

(c) Third line : Number of inputs to be read for the test case.(nI)

(d) Fourth line : Number of queries corresponding to the test case.(nQ)

(e) Next 'nI' lines contain elements of the test case

- Format : ¡address of the element¿ ¡value of the element¿ (Both items are separated by space)

(f) Next 'nQ' lines contains Query for the test case
Format of the query :

- Load_MH : 0
- Load_HT : 1
- Access(key) : 2 <key>
- Display : 3

(g) (b,c,d,e,f) steps are repeated 'nT' times

Example
Each enumerated element is a line in input file

(A) 2

(B) 2

(C) 3

(D) 5

(E) 25 30

(F) 20 18

(G) 11 15

(H) 2 20

(I) 2 11

(J) 3

(K) 1

(L) 3

(M) 1

(N) 2

(O) 3

(P) 15 10

(Q) 16 11

(R) 2 15

(S) 1

(T) 2 15

For the example above, The input file contains line from A-T

- (A) : Total number of test cases = 2
- (B) : Hash Table size for 1st test case = 2
- (C) : Number of inputs for 1st test case = 3
- (D) : Number of Queries for 1st test case = 5
- (E) - (G) : Inputs for 1st test case in format <address> and <value>

- (H) - (L) : Queries for 1st test case.
  - (H) : Access element with key 20
  - (I) : Access element with key 11
  - (J) : Display
  - (K) : Load_HT
  - (L) : Display
- (M) : Hash Table size for 2nd test case = 1
- (N) : Number of Inputs for 2nd test case = 2
- (O) : Number of queries for 2nd test case = 2
- (P) - (Q) : Inputs for 2nd test case in format ¡address¿ and ¡value¿.
- (R) - (T) : Queries for 2nd test case

Output : You should output a file hw1_q2_out.txt.
The contents of the output file should be answers to each query in the input file.
Output of Query Format :

- Access(key) : (0/1/-1) ¡value of the element¿
  - 2 numbers separated by a space : 1st number corresponds to whether the element is in HT/MH and second number corresponds to value of the element
- Display : Two lines w.r.t each display query
  - First line corresponds to elements in HT. Elements should be space separated.
  - Second line corresponds to elements in MH. Elements should be space separated.
- Output of each test case should be separated by an empty line.
- Note : Load_HT, Load_MH has no outputs

Example:
For the inputs shown above, The output file should contain following lines (I,II,..IX are just line numbers, output file shouldn't have these line numbers)

I 0 18

II 1 15

III 20 25

IV 11

V 20 11

VI 25

VII

VIII 1 10

IX 0 10

Explanation:

- I : corresponds to query in line (H)
- II : corresponds to query in line (I)
- III, IV : corresponds to query in line (J)
- V,VI : corresponds to query in line (L)
- VII : Empty line to signify end of test case 1
- VIII : corresponds to query in line (R)
- IX : corresponds to query in line (T)

What to turn in: Your commented code, a short report explaining what you did, as well as the output file that your code generates `hw1_q2_out1.txt`.