CISC-235*
Test #3
March 23, 2017

Student Number (Required) ______________________

SOLUTIONS

Name (Optional) ________________________________

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but will not be re-marked under any circumstances.

The test will be marked out of 50.

| Question 1     | /12 |
| Question 2     | /12 |
| Question 3     | /12 |
| Question 4     | /12 |
| Question 5     | /2  |
| **TOTAL**      | /50 |

“We all want to forget something, so we tell stories”

Happy birthday to Akira Kurosawa
Question 1 (12 marks):

Create a hashing function for playing cards. Each card is represented by a pair \((X,Y)\) where \(X\) is a number in the range 1 to 13, and a \(Y\) is a letter which is either “C”, “S”, “H”, or “D”. You can use this pair as the key.

a) [4 marks] Create your hashing function and explain your reasoning.

One possible solution: use the built-in “ord” function to convert \(Y\) to an integer then multiply this by 100 and add \(X\), i.e. \(h(X,Y) = X + 100 \times \text{ord}(Y)\).

This combines both \(X\) and \(Y\) without any “overlap” - each \((X,Y)\) pair has an unique \(h(X,Y)\) value. This avoids any forced collisions resulting from two items getting the same hash value.

Marking: there are infinitely many good solutions. Others might include 
\(h(X,Y) = X \times \text{ord}(Y)\)
or
\(h(X,Y) = \text{floor}(m \times \{\text{fractional part of } (X \times \text{ord}(Y) \times V) \})\) where \(V\) is a fixed number between 0 and 1
etc.

Students might propose a mapping function \(f()\) for \(Y\) such that (for example) \(f(“C”) = 1, f(“S”) = 2, f(“H”) = 3, f(“D”) = 4\) instead of using \(\text{ord}()\), and then compute something like \(h(X,Y) = 10X + f(Y)\) ... this also produces distinct hash values for all cards.

The important idea is that the student must give a function (or functions, if they decide to use double hashing) that utilizes both parts of the key, and does not create unnecessary collisions by producing the same hash value for different cards.

As an example of a function that is not very good, summing the digits of \(X\) and \(\text{ord}(Y)\) is guaranteed to create some duplicate hash values since \(X=4\) and \(X=13\)
will give identical results when combined with \text{ord}(Y) in this way.

b) [4 marks] How will you handle collisions? Explain your choice.

One solution: I would choose linear probing. The set of possible keys is so small that it is easy to allocate a table large enough to make collisions unlikely. With few collisions, there is no real need for a sophisticated conflict resolution method.

However, any of the methods we have studied would also be effective and acceptable. For example students might decide to use double hashing, with \( h'(\) taking \( X \) and \( Y \) as its parameters and \( h''(\) also taking \( X \) and \( Y \) as its parameters. Note that double hashing where \( h'(\) takes only \( X \) as its parameter and \( h''(\) takes only \( Y \) as its parameter is not a good solution: all cards will initially map to a set of just 13 addresses, which creates unnecessary collisions.

Marking: As long as the student chooses a collision resolution and gives a justification for that choice, please give 4 /4

If the student gives a collision resolution method but does not explain why they chose that, please give 2/4

c) [4 marks] What criteria would you use when choosing the size of your hash table?

One solution: I would consider the number of cards to be stored – for example, if the structure is to be used to store a player’s hand in traditional poker then we only need to store 5 cards. A load factor of 1/2 is generous, so I would make the size of table about twice the number of cards to be stored. Given my choice of hash function and collision resolution, I would choose a prime number for the table size. If there is a performance goal that must be met (such as “no probe sequence is allowed to look at more than 3 locations”) I would choose a table size that meets this requirement.
Marking: The main criteria are the ones mentioned:
  - number of values to be stored
  - special constraints on table size imposed by the hashing function and collision resolution method chosen
  - performance goals

Students who come up with at least two of these should get 4/4. Students may come up with other criteria – please use your judgment as to their validity.

Students who give at least two valid criteria should get 4/4, even if their criteria do not match mine.

Students who only come up with one criterion should get 2/4.
Question 2 (12 marks):

Here is a collision resolution method that combines chaining with open addressing. In this method we resolve collisions with chaining, but each chain is allowed to contain at most 2 values. If we try to insert a value into a location where the chain is full, we use linear probing to find the next address where the chain has length < 2.

a) [4 marks] Show the result of inserting these values:
   3, 23, 14, 6, 13, 24, 25, 17, 4
   into this hash table using h(k) = k % 10

Solution:

```
0
1
2
3  3  23
4  14 13
5  24 25
6  6  25
7  17
8
9
```

Marking: If they decide to do inserts at the head of the chains instead of at the tails, each chain would be reversed.

If they clearly know how it is supposed to work but they make one error of placement, give full marks. If they make several unrelated errors, give 3/4 or 2/4 depending on whether they are errors of arithmetic (3/4) or errors of comprehension (2/4)

(This question continues on the next page.)
b) [4 marks] How would you handle the “delete” operation, so as to guarantee that “search” would function properly?

We have to replace the deleted value in the chain with “Deleted”, because later we might search for a value that was pushed to a later chain due to collisions. If we just delete the chain node containing the value to be removed, “search” could end prematurely when it reaches a chain with < 2 elements.

Marking: if the student suggests just deleting the node containing the value, rather than replacing it with a “Deleted” flag, give 1/4

If the student has the right idea but makes a mistake in explaining it, give 2/4 or 3/4 depending on the severity of the error.

c) [4 marks] Compared to standard open addressing with linear probing, would you expect this method to have more primary clustering, less primary clustering, or about the same amount of primary clustering? Explain your answer.

Solution: The amount of primary clustering should be reduced because collisions do not spill over into other parts of the table as quickly as with standard open addressing with linear probing. In the example above, we can see that if the next insertion hashes to addresses 3, 4, 5, 6, or 7 the element will end up in 7’s chain rather than spill over to 8. We can expect that it will take
longer before big blocks of consecutive filled locations (i.e. primary clusters) appear. However it is still true that some amount of primary clustering can occur: the probability that the next insertion will go into 7’s chain is much higher than the probability that it will go anywhere else.

Marking: Students may correctly observe that the chains of length 2 are conceptually identical to inserting an extra array location after each existing location, effectively just doubling the size of the array, with the proviso that no key will be directly hashed to any of these new locations. From this it is reasonable to conclude that primary clustering can still exist but that it will be reduced since the extra locations serve as “buffers” between the addresses that items will be hashed onto.

If the student claims there will be more primary clustering, give 1 / 4

If the student claims there will be the same amount of primary clustering, give 2/4
Question 3 (12 marks):

Suppose you are implementing a priority queue using a Max-heap. How would you make sure that when two items have the same priority, the one that was added earlier gets to the head of the queue earlier? You may assume there is a built-in function `clock()` that returns the current system time.

Explain the modifications (if any) that you would need to make to the standard `insert()` and `remove_largest()` algorithms.

Solution: We can use the `clock()` function to attach a time stamp to every item that we add to the queue. At any point in our Max-heap operations where we are dealing with two items of equal priority, we can make sure the one with the earlier time stamp ends up as the parent of the other.

Modifications: When we perform an insert, as we bubble the item up the heap, we only swap it upwards if its priority is strictly less than its parent’s priority. If they are equal, the new item must remain below the parent because the parent has been in the heap for some amount of time already.

For `remove_largest`, when we move the bottom element to the root and start to bubble it down, any time the larger child is an item with the same priority as the one we are moving, we compare the time stamps. The one with the earlier time stamp must be the parent of the one with the later time stamp.

When we are comparing the two children of a vertex to see which is the larger, if they have the same priority then we must choose the one with the earlier time stamp as the largest child.

Marking: If the student demonstrates that they understand how Max-heaps work but can’t answer the question, please give at least 6/12.

If they get the idea the `clock()` function can be used to distinguish between items with the same priority, please give at least 7/12.
Correct modifications for insert() are worth 2 marks

Correct modifications for remove_largest() are worth 3 marks

There may be other ways to solve this problem – please evaluate alternative answers carefully – a solution that “almost works” should get about 9 or 10.
Question 4 (12 marks):

In a connected unweighted graph, the distance between two vertices is the number of edges in the shortest path that joins them. For example in this graph the distance between C and E is 3, and the distance between D and F is 2.

Explain how we can use multiple executions of Breadth First Search to compute the distance between all pairs of vertices in a connected unweighted graph.

The next page is available for use in answering this question.
Solution:

Create a matrix $D$ to hold all the distances: $D[i,j]$ will hold the distance from vertex $i$ to vertex $j$. Initialize $D[i,j] = \infty$ for all $i$ and $j$.

for each vertex $i$:

Use BFS($i$) to construct a search tree beginning at vertex $i$.
As each vertex $j$ is assigned to a level in the search tree,
if the level is $< D[i,j]$:

$D[i,j] = \text{level}$
$D[j,i] = \text{level}$

Marking:

If the student shows that they understand BFS but cannot get any further than that, please give at least 6/12.

If they understand that BFS gives the distances from the start vertex to all other vertices in the graph, please give at least 8/12.

If they have the idea of using every vertex as the starting point for a BFS, please give at least 10/12.
Question 5 (2 marks):

Why can’t we use this for double hashing:

\[ h(k, i) = (h'(k) + \text{randint()*}i) \mod m \]

where \text{randint()} returns a random positive integer every time it is called?

Solution: This would make it impossible to recreate the probe sequence used when inserting a key, so each search and delete would have to examine every address in the table.

Marking: If the student says anything about the results not being predictable or reproducible, give at least 1 mark. For 2 marks they should describe a concrete problem. They may come up with something other than the one I did – if it is a realistic problem, give 2 marks. If it is unrealistic (such as “the probe sequence might look at the same address every time”) please use your judgment to assign either 1 or 2 marks.