CSCE 221 Cover Page
Homework Assignment #3
Due April 25 at 23:59 pm to eCampus

Hayes Hayes UIN: 525003952

User Name: AsaHayes E-mail address: asahayes@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website: `http://aggiehonor.tamu.edu/`

Type of sources				
People				
Web pages (provide URL)				
Printed material				
Other Sources				

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Asa R Hayes Date

Homework 3 (100 points)

due April 25 at 11:59 pm to eCampus.

Write clearly and give full explanations to solutions for all the problems. Show all steps of your work.

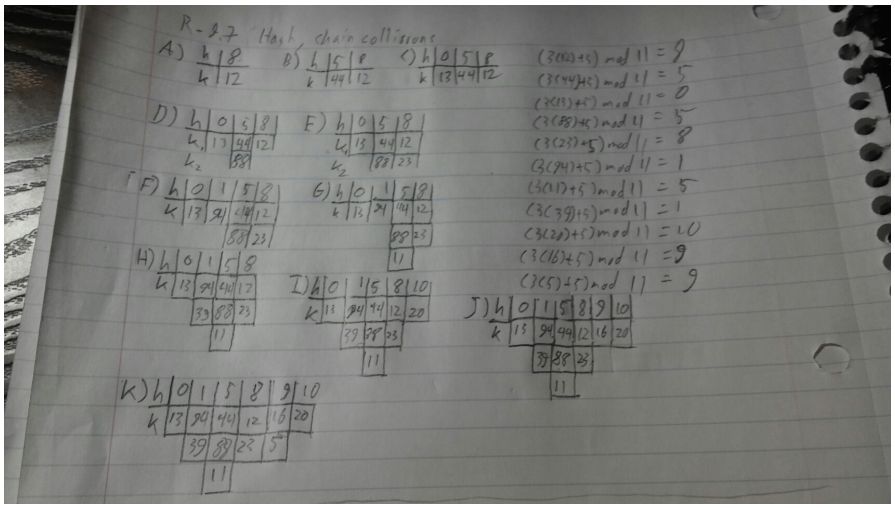
Reading assignment.

- Hash Tables Chap. 9
- Heap and Priority Queue, Chap. 8
- Graphs, Chap. 13

Problems.

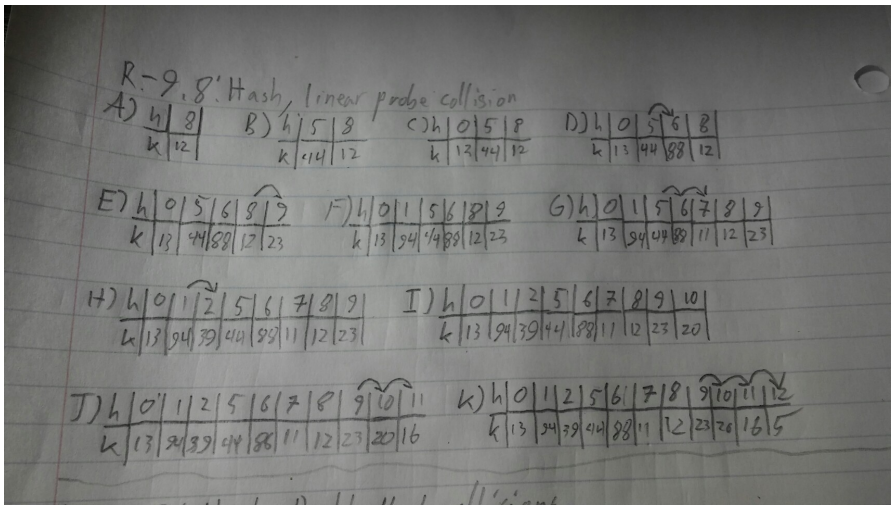
1. (10 points) R-9.7 p. 417

   Draw the 11-entry hash table that results from using the has function, $h(k) = (3k + 5) \bmod 11$, to hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, assuming collisions are handled by chaining.



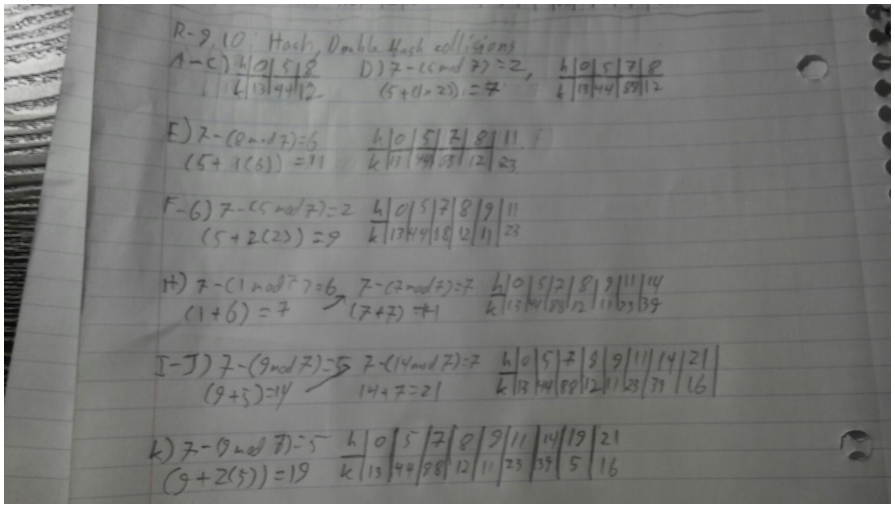
2. (10 points) R-9.8 p. 417

   What is the result of the previous exercise, assuming collisions are handled by linear probing?



3. (10 points) R-9.10 p. 417

What is the result of Exercise R-9.7, when collisions are handled by double hashing using the secondary hash function $h_s(k) = 7 - (k \bmod 7)$?



4. (10 points) R-8.7 p. 361

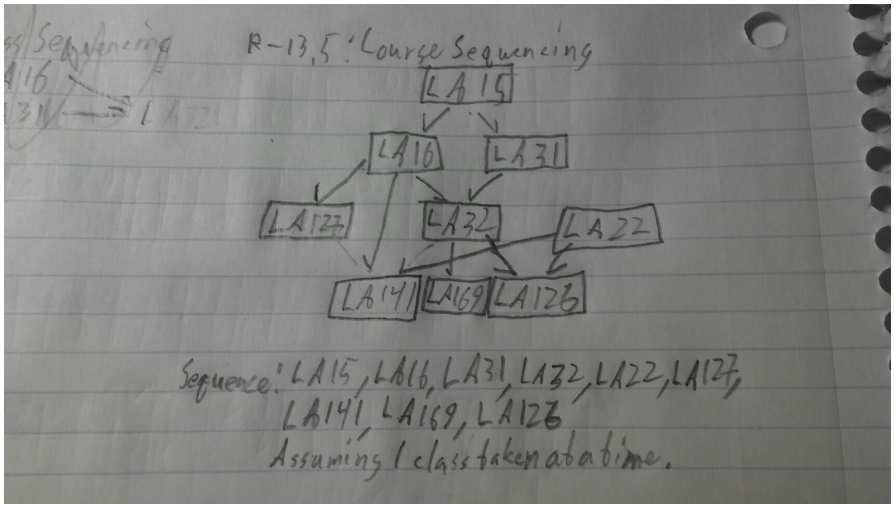
An airport is developing a computer simulation of air-traffic control that handles events such as landings and takeoffs. Each event has a *time-stamp* that denotes the time when the event occurs. The simulation program needs to efficiently perform the following two fundamental operations:

- Insert an event with a given time-stamp (that is, add a future event)
- Extract the event with smallest time-stamp (that is, determine the next event to process)

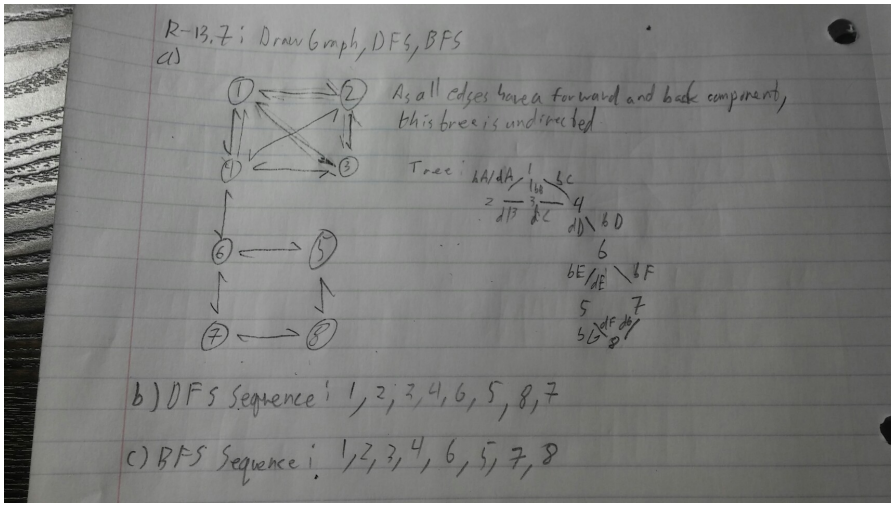
Which data structure should be used for the above operations? Why? Provide big-oh asymptotic notation for each operation.

Because we only need to access elements (time-stamps) in the front of the timeline, we can narrow down the options for a good structure into some generally faster ones. For insertion, since we have to insert at a specific time, this complicates things. We can't use stack or queue because those can't insert or access midway into the timeline. A binary tree might work, but every push and pop operation would require a large amount of rebalancing. Heaps would have less balancing issues, but also insufficient ordering for a sequence of times. The best course for this application would be a linked list, as it has all the features needed: sequential ordering, ability to remove smallest element, and ability to insert between two values. Collision shouldn't be an issue, as you can't schedule two planes to be coming in at the same due to physical collision, so you would just need a different list for each runway and gate which would prevent conflicts from occurring. For a linked list, a remove_front operation would be O(1), while the insert would be O(n) (O(n) for the search, O(1) for the insertion).

5. (10 points) R-13.5, p. 654



6. (10 points) R-13.7, p. 655



7. (10 points) R-13.16, p. 656

   Modifying Dijkstra's algorithm to directly output a tree would require a couple of changes besides formatting for output. First, you would have to give each vertex a new member variable which would need to be made to point to the node that it is closest to, which will be updated every time the relax function comes back positive. We can then add the edge from a vertex to the node closest to it into the tree when it gets removed from the algorithm's priority queue.

8. (10 points) R-13.31, p. 657

   For a complete undirected graph, the minimum distance between any two nodes is 1, so the DFS tree would only have a depth of one. There would be the root, and then every other node one level down from it. All nodes would be cross-linked with each other and back-linked with the root. The tree would be this way for any node as the root.

9. (10 points) C-13.10, p. 658

   To do get an Euler tour, you would first loop thru the graph and add all of the edges onto a stack. Once all of the edges of the graph were in the stack, you would then run a DFS, removing vertices from the stack as you go. If at the end the stack is empty and you are back at root (which is always the case at the end of a DFS), then you have constructed an Euler Tour.

10. (10 points) C-13.15, p. 659