

# CS 14 - Summer 2003 - Quiz 4

July 23, 2003

## 1 True/False

Circle *T* or *F* as appropriate.

1. **T F** Hash tables using linear probing to resolve collisions have a worst case *search* time of  $O(n)$ , the number of items in the table.
2. **T F** Hash tables using quadratic probing to resolve collisions have a worst case *search* time of  $O(n)$ , the number of items in the table.
3. **T F** If a hash is initially allocated small and expands by doubling when nearing the acceptable capacity threshold, the *average – case* time for insertion is  $O(n)$
4. **T F** If a hash is initially allocated small and expands by doubling when nearing the acceptable capacity threshold, the *worst – case* time for insertion is  $O(n)$
5. **T F** A good hash function should produce values that are all prime.
6. **T F** A graph consists of a set of *vertices*, and a set of *edges* connecting those, and directions for how to place the vertices.
7. **T F** A graph is *connected* if every pair of vertices have an *edge* between them.
8. **T F** Order is important for the nodes in the linked-lists in a list-representation of a graph.

## 2 Multiple Choice

Circle the letter of the correct answer. *Running-time* questions are all in terms of input size  $n$

1. Which of the following is *not* an important feature of a good hash function:
  - (a) Uniform output distribution
  - (b) Good “spread” (nearby inputs get distributed far apart)
  - (c) Random output
  - (d) Fast running time

2. A binary search tree can be thought of as a \_\_\_ with a value at each node
  - (a) Directed, weighted, acyclic<sup>1</sup> graph
  - (b) Undirected weighted graph
  - (c) Directed, unweighted, acyclic graph
  - (d) None of the above
3. An adjacency matrix for  $n$  nodes
  - (a) Uses  $O(n^2)$  space and gives  $O(1)$  lookup to find edges
  - (b) Uses  $O(n)$  space and gives  $O(1)$  lookup to find edges
  - (c) Uses  $O(n^2)$  space and takes  $O(n)$  lookup to find edges
  - (d) Uses  $O(n)$  space and takes  $O(n)$  lookup to find edges

### 3 Short Answer

*If you want the option to request a regrade available, your answers to these questions must be made in pen.*

1. A “perfect hash function” is a special kind of hash that can be created when you know ahead of time exactly which keys are going to be inserted in the table. Describe in English why a perfect hash is useful. (Think about it for a while and take your time. A correct answer to this should be about 2-4 sentences.)

---

<sup>1</sup>No cycles

2. A graph representing the prerequisites for the CS courses here at UCR could have each course as a vertex and a directed edge from node  $a$  to node  $b$  if  $a$  is a prereq for  $b$ . (Thus, anyone can take a course with no incoming edges in this graph.)
- (a) Given the following list of prereqs, draw the appropriate graph:
- CS 10 comes before CS 12
  - CS 12 comes before CS 14
  - CS 10 comes before CS 61
  - CS 14 comes before CS 141
  - CS 141 comes before CS 100
  - CS 20 has no prereqs and is not required for any other course.
- (b) Which of the following terms apply to the graph you constructed (mark yes or no for each)
- Directed
  - Acyclic
  - Connected
  - Complete
- (c) Is it reasonable to have a graph representing prereqs containing a cycle? Describe why or why not.
- (d) Is it reasonable to have a graph representing prereqs that is connected? Describe why or why not.

3. Write `bool hashInsert(pair<string, bool>* table, int tableSize, string toInsert)`, given a hash function `unsigned int hashString(string toHash)`. You may assume that `tableSize` is much larger than the number of elements that will be inserted. Your function should return true unless `toInsert` already exists in the table. Use **quadratic probing**. (Note that the `bool` values in `table` represent the “dirty-bits”, and that the only `pair` operations you need are `table[i].first` and `table[i].second`.)