Final Study Guide

Hashing

1. Match each of the following terms with their corresponding definitions:

|  |  |
| --- | --- |
| Term | Definition |
| Hash Code | The percent of used spots in a Hash Table which, if exceeded, results in a resize of the Hash Table. |
| Hash Table | A collision resolution strategy where we look to adjacent spots in a Hash Table if a collision occurs. |
| Collision | An array that stores elements using Hash Codes. |
| Load Factor | This occurs when two elements would be placed in the same location in a Hash Table, given their hash codes. |
| Linear/Quadratic Probing | A number or string that is used to represent an object. This value represents a summarization of the object and is often called a “digest.” |
| Chaining | A collision resolution strategy where linked lists are used to hold elements at every position in a Hash Table. When a collision occurs, the new element is added to a linked list at the position determined by the element’s hash code. |

1. Given the following Hash Table and a load factor of 50%:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| B | A |  |  |  | C |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Add each of the following elements to the Hash Table above:

|  |  |
| --- | --- |
| Element | Hash Code |
| A | 9 |
| B | 16 |
| C | 13 |
| D | 15 |
| E | 12 |
| F | 7 |
| G | 11 |
| H | 24 |
| I | 31 |
| J | 13 |
| K | 15 |

*Note: You should resolve any collisions using Linear Probing.*

*Note: You should show the steps of each add() operation. This includes how you arrived at a location for each new element as well as the state of the Hash Table after each call to add().*

1. Explain why the following rule exists: “if you override equals(), then you must also override hashcode().”
2. Suppose you have the following Hash Table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Giraffe** | **Koala** | **Bear** | **Antelope** | null | null | null | **Platypus** | null | null |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Assuming the element “Bear” has a hash code of 160, explain the steps involved when calling remove(“Bear”).

1. It is important not to replace an element in a Hash Table with a null value when remove() is called. For example, the method call from (4) should not result in the following table:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Giraffe** | **Koala** | null | **Antelope** | null | null | null | **Platypus** | null | null |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Explain why this is an important feature of a hash table class.

1. Give three strategies that help to prevent collisions in a Hash Table.
2. What are the advantages and disadvantages of a high load factor?
3. Conversely, what are the advantages and disadvantages of a low load factor?
4. Given a relatively small number of collisions in a hash table, give a big-oh estimate of the following operations on a hashing structure:
   1. insert(x):
   2. remove(x):
   3. contains(x):
   4. size():
5. Given repeated collisions in a hash table, give a big-oh estimate of the following operations on a hashing structure according to the worst-case scenario:
   1. insert(x):
   2. remove(x):
   3. contains(x):
   4. size():

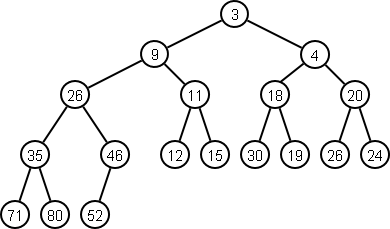
## Heaps

1. How do priority queues differ from regular queues?
2. What three methods are part of the priority queue ADT?
3. State the heap ordering property. Be as clear as possible when answering this question.
4. What is a complete binary tree?
5. A complete binary tree can be stored efficiently in an array. Given an index i in this array, how would you identify the index of a parent node, left child or right child node assuming:
   1. The root of the tree is located at index one
   2. The root of the tree is located at index zero
6. Draw the resulting heap after adding the following elements from left-to-right: 16, 13, 7, 12, 4, 0, -2.

*Note: You should show the heap in tree-form after each add() operation above.*

*Note: Make sure to denote clearly each percolateUp() operation and how it was executed.*

1. For the following heap, show the steps involved when calling deleteMin().



*Note: Your solution should clearly show each step of the percolateDown() operation.*

1. What is the difference between a min-heap and a max-heap?
2. Name three algorithms we have discussed this quarter that use a heap as part of their solution.

## Sorting

1. What is an **inversion**?
2. What is the worst case running time of the following sorts.
   1. bubbleSort()
   2. selectionSort()
   3. insertionSort()
3. Write a method that performs the selection sort algorithm on an input array.
4. Write a similar method for insertion sort.
5. Which of the sorts from (2) have the best running time in the best case scenario? What is the big-oh in the best case?
6. Given the sort chosen from (5), what type of input is necessary to get the best case scenario?
7. Insertion sort as an algorithm has weaknesses depending on the type of input given to the algorithm.
   1. Explain how shell sort is an improvement over insertion sort.
   2. How does shell sort address the weakness of the insertion sort algorithm?
8. Merge sort and quick sort are called divide-and-conquer algorithms. Explain how divide and conquer algorithms work.
9. Show the recursive calls and merge steps for the merge sort algorithm on the following input: [99, 22, 117, 41, 98, 110, 1, 22].
10. For the merge sort algorithm:
    1. Is it possible to get stack overflow for any input array?
    2. Does the algorithm use extra space to perform the sort?
    3. What is the worst case AND average case big-oh?
11. Show the steps of the quick sort algorithm on the following input: [6, 5, 12, -4, 33, 18, 2, 3, -6, 10, 3, 1]

*Note: Use the first element as your pivot for each step of the algorithm.*

*Note: Make sure to clearly denote your parititioning step, with indices i and j (as shown in class).*

*Note: You can declare that your work uses a well known sort (such as insertionSort()) on any sub-arrays of size three or smaller.*

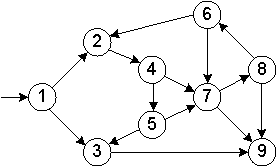
1. Explain how median-of-three partitioning works for pivot selection.
2. Give an example input array where median-of-three does not adequately partition the array.
3. For the quick sort algorithm:
   1. Is it possible to get stack overflow for any input array?
   2. Does the algorithm use extra space to perform the sort?
   3. What is the worst case AND average case big-oh?
4. What does it mean that heap sort can be programmed as an **in-place sort**?
5. Explain how the in-place heap sort routine works.
6. What is the minimum amount of work necessary for any comparison based sort. ie. given a comparison sort f(n), give a g(n) such that f(n) = Ω(g(n)).

## Graphs

1. Match each of the following terms with their corresponding definitions:

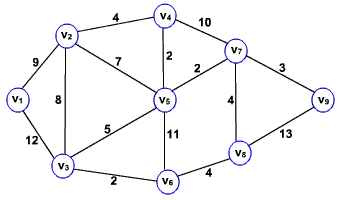
|  |  |
| --- | --- |
| Term | Definition |
| Vertex | A graph that includes a value on each edge. These values can represent distance, cost or some other numerical quantity. |
| Edge | A graph where for every pair of vertices u and v, there is a path from u to v. |
| Directed graph | A linear sequence of edges in a graph that allow you to traverse from a source node to a destination node. |
| Weighted graph | A path that begins and ends at the same vertex. |
| Path | Given a vertex u, all edges (u, v) from u to another vertex v are said to be \_\_\_\_\_\_\_\_\_\_\_\_ to u. |
| Cycle | The number of incoming edges to a vertex. |
| In-degree | The elements we store in a graph structure, along with their relationships. |
| Out-degree | Represents a relationship between two vertices in a graph. |
| Sub-graph | Given a graph G = (V, E), this type of graph contains a subset of V and E. |
| Complete graph | Two vertices u and v are said to be \_\_\_\_\_\_\_\_\_ if there exists an edge (u, v). |
| Adjacent | The number of outgoing edges from a vertex. |
| Incident | A graph where the edge (u, v) is not the same as (v, u). |
| Connected graph | A graph that contains an edge between every pair of vertices. |

1. What is a **DAG**?
2. Explain how adjacency lists are used to store vertices and edges in a graph structure. Be specific.
3. Which of the following graph representations uses more memory in the worst case? Justify your answer.
   1. Adjacency list
   2. Adjacency matrix
4. Which of the following graph representations uses more memory in the best case? Justify your answer.
   1. Adjacency list
   2. Adjacency matrix
5. Given the following graph representations, what is the runtime cost of searching for an edge (u, v) in a graph?
   1. Adjacency list
   2. Adjacency matrix
6. Show the depth-first search for the following graph:



*Note: Make sure to show each step of the algorithm, including any backtracking that occurs.*

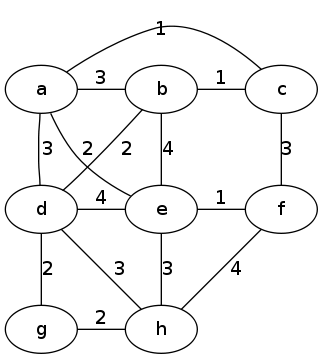
1. For the following graph, show the shortest path from V9 to all other nodes using Dijkstra's algorithm:



*Note: Your solution should show each update performed during the algorithm.*

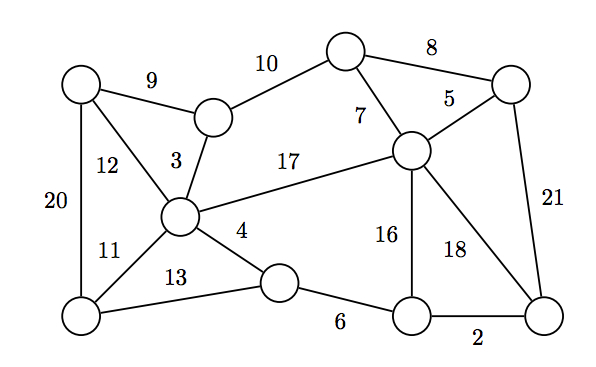
*Note: Your answer should include the shortest path cost for each vertex Vi as well as the path leading from V9 to Vi.*

1. What type of structure would we typically use to store vertices and their current distance values as part of Dijkstra's algorithm?
2. When updating vertex distances during Dijkstra's algorithm, how would you update the position of a vertex in the structure from (7).
3. Is each shortest path generated by Dijkstra's algorithm unique? ie. given a shortest path from u to v, can another path be found in the same graph with the same total cost as the shortest path found by Dijkstra's?
4. Show a minimum spanning tree for the graph below using Prim's Algorithm:



*Note: Your solution should show the set of known and unknown vertices for each step of Prim's algorithm.*

1. Show a minimum spanning tree for the graph below using Kruskal's Algorithm:



*Note: You do not need to show the steps of the union-find algorithm. Instead show how each edge is chosen as part of the algorithm.*

1. Is the minimum spanning tree in a graph unique always unique? Explain when this will or will not be the case.

## Algorithm Design Techniques

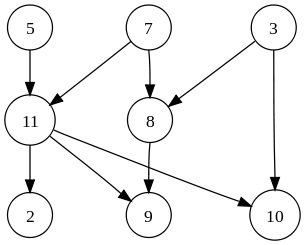
1. For each of the following categories of algorithms, give a description of how that technique is used to solve a problem:

|  |  |
| --- | --- |
| Design Technique | Description |
| Brute Force Algorithm |  |
| Greedy Algorithm |  |
| Decrease and Conquer Algorithm |  |
| Divide and Conquer Algorithm |  |
| Transform and Conquer Algorithm |  |
| Dynamic Programming |  |
| Backtracking Algorithm |  |

1. For each of the following categories, give two example algorithms we have studied this quarter that use the technique:

|  |  |
| --- | --- |
| Design Technique | Algorithms |
| Brute Force Algorithm |  |
| Greedy Algorithm |  |
| Decrease and Conquer Algorithm |  |
| Divide and Conquer Algorithm |  |
| Transform and Conquer Algorithm |  |
| Dynamic Programming |  |
| Backtracking Algorithm |  |

1. State the definition of the travelling salesman problem.
2. Give a big-oh estimate of the brute force solution to the travelling salesman problem. Justify your answer.
3. Give a description of a greedy algorithm that, given coin denominations 1, 5, 10, 25, will determine the minimum number of coins needed to make N cents.
4. Show the steps of the topological sort algorithm on the following graph:



1. Explain why it is necessary for a graph to be a DAG before performing topological sort.
2. Explain how you would use a priority queue (heap) as part of the topological sort algorithm. How are the keys sorted in the heap?
3. Name two types of problems that, when represented with a graph, could benefit from the results of topological sort.
4. Given an array of size N, give an algorithm that uses dynamic programming to determine the minimum number of coins it takes to make N cents with denominations C1,C2, C3, … ,Cm.

*Note: You may not use any extra space beyond the array of size N.*

*Note: List the steps of your algorithm clearly, or give psuedocode as an answer.*

1. State the definition of the eight-queens problem.
2. Explain how an algorithm can use recursive backtracking to solve the eight-queens problem.