Final Exam Study Guide

(If you know everything on this study guide, you should do well on the final exam.)

Pre-Midterm Assumptions:

The final exam is comprehensive, so you are still expected to know the material we covered up through the mid-term. REFER TO THE MIDTERM EXAM STUDY GUIDE for any other material you are expected to know that isn't explicitly mentioned here.

General Guidelines:

We have studied many ***data structures*** this semester. In general, for each data structure, you should know the following:

- What an instance of the data structure looks like (e.g., how a B-Tree is organized)
- What the rules/properties are for that data structure (e.g., the properties that describe a valid B-tree, or the structure and heap-order properties of a heap).
- The main operations that are applied to that data structure, as part of the interface (e.g., "insert" and "deleteMin" are the primary operations of a priority queue)
- A typical implementation of these operations.
- The runtime requirements of the operations applied to that data structure (e.g., "deleteMin" on a priority queue takes time O(log n))
- Given an example data structure (with actual data), show the result of applying a particular operation on that data (e.g., show the result of inserting a new element in a given B-tree).

We have studied many ***algorithms*** this semester. In general, for each algorithm, you should know the following:

- How the algorithm works.
- Given an example, show how the algorithm would process that example (e.g., show the result of one pass of Shellsort using an increment of 5 on a particular set of input numbers).
- Know the data structures that would be used to produce the most efficient implementation of the algorithm (for example, a priority queue is used in Dijkstra's algorithm to determine the next vertex to process, based on the smallest d_v value so far)
- Know the runtime of the algorithm.

Additionally:

- Given a problem to solve (e.g., "give pseudocode to generate a maze of arbitrary size"), you should be able to decide which algorithm (and data structure) is most appropriate (i.e., efficient) for that problem, and generate pseudocode to solve it.
- Given an algorithm (or pseudocode), you should be able to analyze the computational complexity of that algorithm ("big-O" notation).
- You should understand the basic theory of algorithmic analysis (O, o, Θ, ω, Ω).

Data Structures:

These are the data structures you are expected to know, per the above general guidelines:

- Arrays
- Linked lists
- Red-Black trees (Know basic structure, properties, and runtime requirements of operations. You do not need to know how the operations are implemented).
- Stacks
- Queues
- B-Trees

- Hash tables
- Extendible hash tables
- Priority queues (heaps)
- Disjoint sets

Additionally, you should know how to implement these in C++, using STL where appropriate/relevant.

Algorithms:

These are the algorithms you are expected to know, per the above general guidelines:

- Internal sorting:
  - Insertion sort
  - Shellsort
  - Heapsort
  - Mergesort
  - Quicksort
  - Bucket sort
- External sorting:
  - Multiway merge
  - Polyphase merge
  - Replacement selection
- Union/find algorithm for disjoint sets
- Graphs:
  - Representations (adjacency list, adjacency matrix)
  - Topological sort
  - Dijkstra's algorithm
  - Shortest path for acyclic graphs (i.e., using topological sorting)
  - Maximum-flow algorithm
  - Prim's algorithm for minimum spanning trees
  - Kruskal's algorithm for minimum spanning trees
  - Depth-first search
  - Breadth-first search
  - Priority-first search
  - Biconnectivity
  - Euler circuits
  - Finding strong components

NP-Completeness:

Know the definitions of P and NP.

Know what is meant by an "undecidable" problem.

Know the relationships between P and NP (i.e., is one a subset of the other, are they equal?), including what is known and what is currently unknown.

Be able to show that a problem is in NP. That is, given a problem, be able to write pseudocode to verify that, given a certificate (i.e., an answer to the problem) that certificate is correct (in polynomial time).

Know what the difference is between an optimization problem and a decision problem.

Given an optimization form of a problem, be able to convert it to a decision problem.

Know the definition of an NP-complete problem.

Know what the formal process is for showing that a new problem is NP-complete.

Know what is meant by a "polynomial time reduction".

Know what the theoretical significance would be of an NP-complete problem being solvable in polynomial time.

For a specific instance of the Hamiltonian Cycle problem, know how to perform the polynomial-time reduction from that instance to an instance of the Traveling Salesman Problem.

For a specific instance of the Clique problem, know how to perform the polynomial-time reduction from that instance to an instance of the Vertex-Cover Problem.

Know what the practical impact is of showing that your real-world application problem is NP-complete (i.e., Can you therefore solve your problem exactly? What alternative do you have for dealing with your problem?)

You do NOT have to know about complexity classes other than P and NP (i.e, you won't be tested on co-NP or PSPACE or EXPTIME)

NP-complete problems we have discussed:

- Hamiltonian cycle problem – HAM-CYCLE = {<G> | G has a Hamiltonian cycle}
- Traveling-salesman problem – TSP = {<G,c,k> | G is a complete graph, c defines the cost of the edges of G, and G has a traveling-salesman tour with a cost at most k, where k is an integer}
- Clique problem – CLIQUE = {<G,k> | G is a graph with a clique of size at least k}
- Vertex-cover problem – VERTEX-COVER = {<G,k> | G is a graph with a vertex cover of size at most k}

Sample Problems

Here are some sample problems to help you prepare for the final exam. These problems are representative of the types of questions you may see on the final. However, these problems do not cover the entire scope of topics you are expected to know; the first 3 pages (as well as the midterm study guide) outline the scope of material you are expected to know. I strongly encourage you to work these problems BEFORE you see the answers. Otherwise, you aren't really testing your own knowledge of the material.

1. External sorting.

You are given a series of integers that are written on an input tape (i.e., T_{a1} below).

Your job is to sort these numbers using 3-way merge (i.e., multiway merge with $k = 3$). Your initial run size is 4 (i.e., M = 4). Show the results after the first 2 passes. (In Pass 2, you do not have to rewrite tape values that did not change from Pass 1. Just show the results on the tapes that change.)

Results after Pass 1:

T_{a1}: 12 7 3 28 74 38 2 5 37 42 22 1 17 45 8 6 51 11 13

T_{a2}:

T_{a3}:

T_{b1}:

T_{b2}:

T_{b3}:

Results after Pass 2:

T_{a1}:

T_{a2}:

T_{a3}:

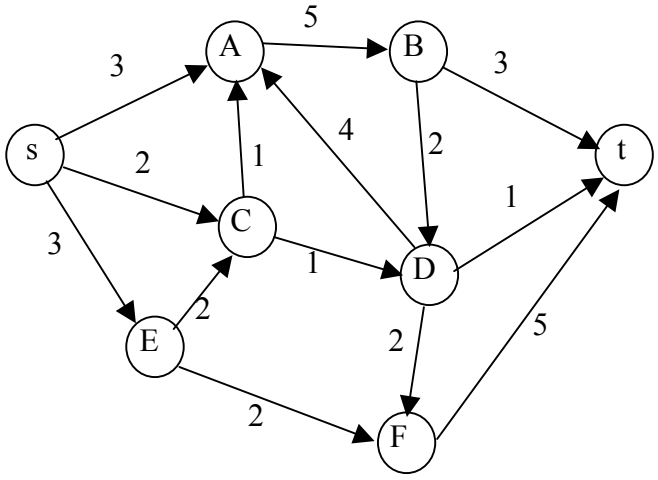
T_{b1}:

T_{b2}:

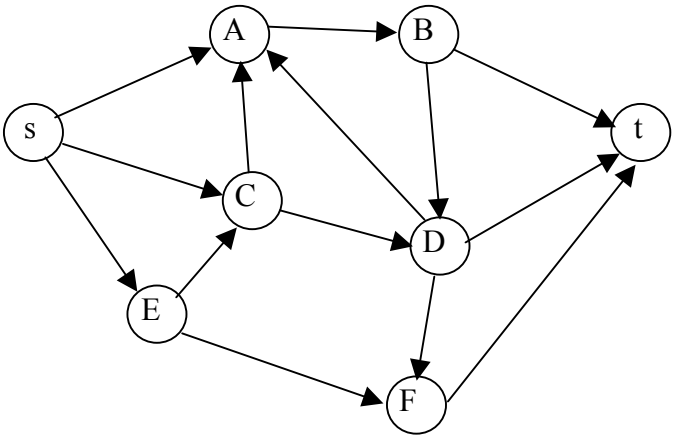
T_{b3}:

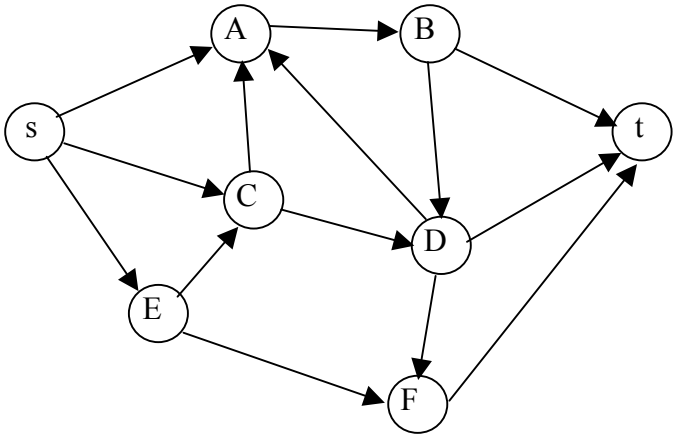
2. Maximum Flow.

Compute the maximum flow for the following graph, giving your answer on the graph template at the bottom of the page. (Additional "scratch graph" templates are provided, which you can use if you want in deriving your answer.)

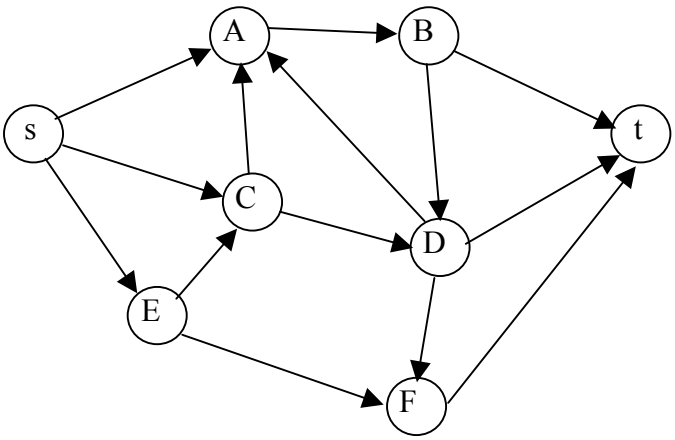


"Scratch graphs", for your convenience if needed:





YOUR MAXIMUM FLOW ANSWER (i.e, label each edge with the flow along that edge that gives the maximum total flow):



3. NP-completeness

Let us define a problem called the SUBSET-SUM(S,*t*) problem. In this problem, you are given a finite set of integers. We'll call this set S. We're also given a "target" integer called *t*. In this problem, we want to know if there is some subset of numbers in S that sums to the value *t*.

For example, let's assume S = {1, 3, 7, 100, 175, 202, 871} and *t* = 205. Then, an answer to this problem is the subset S′ = {3, 202}.

Now, your job is to show that the problem SUBSET-SUM(S,*t*) is in the class NP. You must do this by writing pseudocode that takes an answer to SUBSET-SUM and verifies that it is correct.

Your program's input:

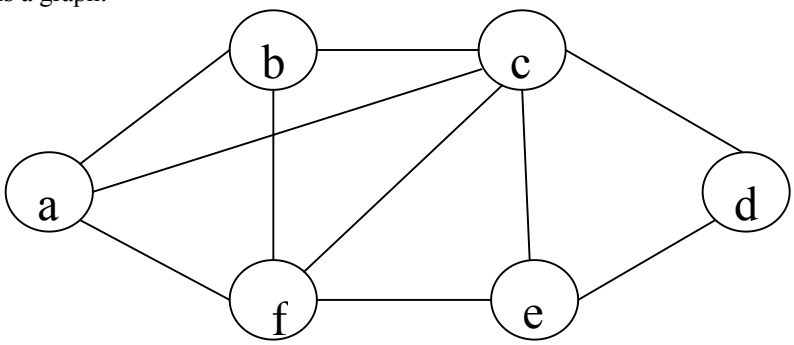
- Set S of integers
- Target value *t*
- Set S′, which you are told is an answer to the SUBSET SUM(S,*t*) problem.

a. Your pseudocode to verify this answer:

b. What is the runtime of your algorithm (in "big-O" notation)?

4. NP-completeness and polynomial-time reductions

Here is a graph:



a. What is the maximum-sized clique for this graph? [You should give the actual clique, not just its size.]

b. State the decision version of the CLIQUE problem (i.e., consistent with how problem statements are made for the theory of NP-completeness).

c. Using the polynomial-time reduction from CLIQUE to VERTEX-COVER that we discussed in class, show the graph G′ that results when you transform the original graph above to an instance of the VERTEX-COVER problem. [You just have to show the graph.]

d. What is the minimum-sized vertex cover for your graph in part c? [You should give the actual vertex cover, not just its size.]

5. What feature of heaps allows them to be efficiently implemented using a partially filled array?

   a. Heaps are binary search trees

   b. Heaps are complete binary trees

   c. Heaps contain only integer data

   d. Every node is smaller than its descendents.

6. Here is an array of integers: 5 3 8 9 1 7 0 2 6 4.

   Write the resulting array after the FIRST iteration of the outer loop of an insertion sort (sorting from smallest to largest):

7. Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning. The resulting array looks like this:

   2 5 1 7 9 12 11 10

   Which statement is correct about the pivot that was used to generate this resulting array?

   a. The pivot could have been either the 7 or the 9.

   b. The pivot could have been the 7, but it is not the 9.

   c. The pivot is not the 7, but it could have been the 9.

   d. Neither the 7 nor the 9 could have been the pivot.

8. Mergesort makes 2 recursive calls. Which statement is true after these recursive calls finish, but before the merge step?

   a. The array elements form a heap.

   b. Elements in each half of the array are sorted amongst themselves.

   c. Elements in the first half of the array are less than or equal to the elements in the second half of the array.

   d. None of the above.

9. If well-implemented, a hash table is O(1) for a lookup, and a binary search tree is O(log n). What is the most significant reason why you might want to use a binary search tree anyway?

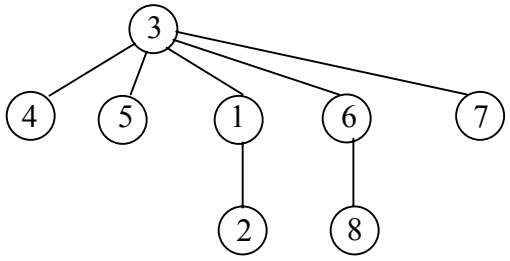
   a. Binary search trees use less memory.

   b. Your hash table might not be big enough.

   c. You want to retrieve your data in sorted order.

   d. Your data consists only of integers.

10. The following tree is the result of a sequence of "union" operations applied to 8 elements that are initially in different sets. Here, unions are performed arbitrarily, without path compression.



   Which of the following is a sequence of unions that could have generated this tree?

   a. union(3,7); union(3,6); union(6,8); union(3,1); union(1,2); union(3,5); union(3,4)

   b. union(4,3); union(5,3); union(2,1); union(2,3); union(8,6); union(8,3); union(7,3)

   c. union(1,3); union(3,4); union(4,5); union(2,5); union(6,8); union(6,2); union(7,6)

   d. None of the above.

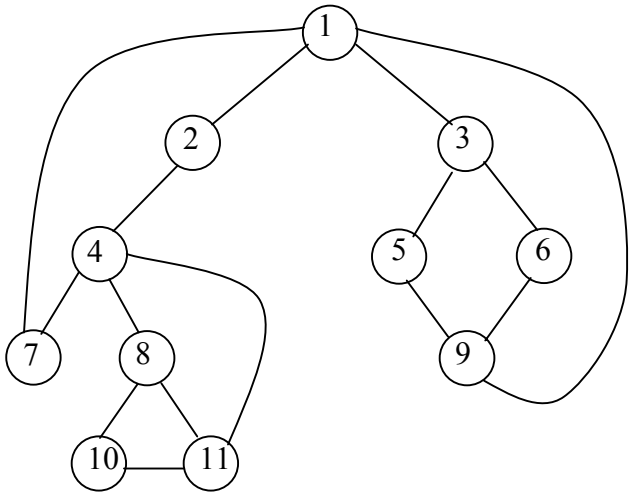
11. For each statement, circle the correct answer.

TRUE FALSE UNKNOWN: The Halting Problem is NP-complete.

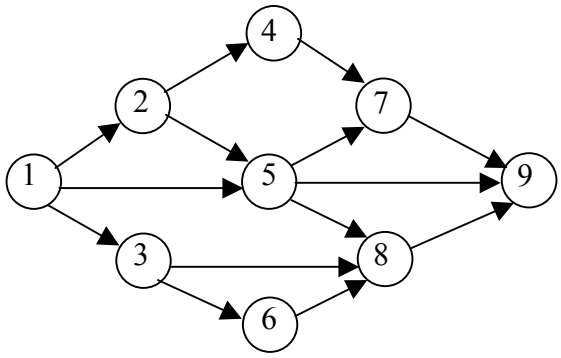
TRUE FALSE UNKNOWN: Traveling Salesman Problem $\in$ P

TRUE FALSE UNKNOWN: Dijkstra's algorithm $\in$ P

12. Circle all of the articulation points in the following graph:



13. Which of the answers below is NOT a valid topological sort of the following graph?



a. 1, 2, 3, 4, 5, 6, 7, 8, 9

b. 1, 3, 2, 6, 5, 4, 8, 7, 9

c. 1, 3, 6, 8, 2, 5, 4, 7, 9

d. 1, 2, 3, 4, 5, 8, 6, 7, 9

14. For each statement, circle the correct answer:

TRUE FALSE Let G = (V, E) be an undirected weighted graph, and let T be a minimum spanning tree of G. If all the edge weights on G are increased by a constant number c, then T is still a minimum spanning tree.

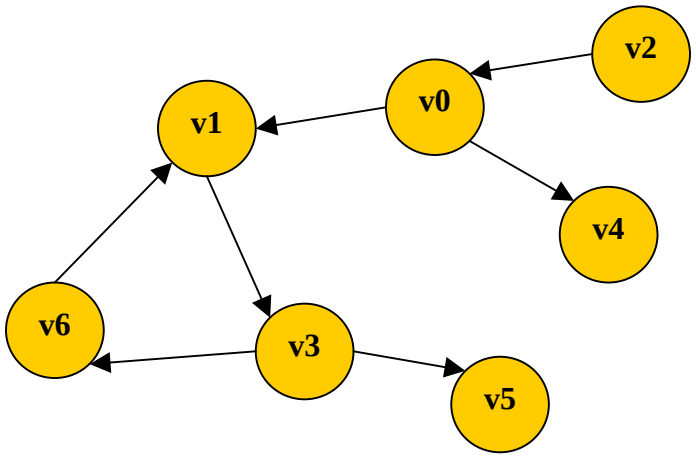
TRUE FALSE An O(V+E) algorithm is known to exist to solve the single source shortest path in a DAG.

TRUE FALSE If $f(n) = \Omega(g(n))$ and $g(n) = O(f(n))$, then $f(n) = \Theta(g(n))$.

TRUE FALSE Red-black trees perform better than binary search trees when performing finds on randomly ordered data.

TRUE FALSE Binary search trees perform better than red-black trees when performing inserts on randomly ordered data.

15. Give the order that the vertices are visited (a) using DFS on this graph, and (b) using BFS (Breadth-first search), starting at v0, with the tie-breaking rule being that vertices are visited alphabetically:



16. How many minimum spanning trees does the following graph have?