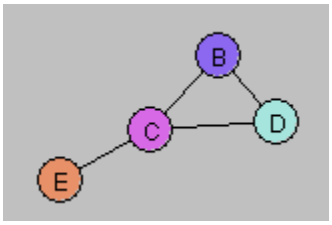
Final Exam A - Solutions
Algorithms, Corazza

I. True/False. [16 points, 2 points each]. *Hint.* Read each of the following carefully before answering. Or, to increase your chance of success, flip a coin. Write answers to Part I in the space provided.

__F__ 1. If *G* is a graph with an odd number of even simple cycles, then *G cannot be* bipartite. **Solution**: Consider a square.

__F__ 2. If *G* is a connected graph with *n* vertices, then *G* must have more than $\frac{(n-1)(n-2)}{2}$ edges. **Solution:** o – o – o – o – o has 5 edges, but $C_{n-1,2} = 6$.

__F__ 3. Suppose *G* is a connected graph with *n* vertices and *n* edges. Then *G* is itself a simple cycle.



__T__ 4. There is no heap with height 11 that has 1000 nodes.
Solution: In a heap of height 11, n must be at least $2^{11} = 2048$.

__F__ 5. If there is an algorithm which runs in exponential time that solves a decision problem Q then Q does *not* belong to *P*. **Solution:** Consider computation of nth Fibonacci number. There is an exponential algorithm that does this, but since there is also a linear algorithm that does it, Fibonacci actually belongs to P.

__T__ 6. Before the discovery of the AKS Primality Test in 2002, all known (deterministic – that is, not probabilistic) solutions to the IsPrime problem ran in superpolynomial time relative to input size.

__T__ 7. Both of the following are true: VertexCover is polynomial reducible to HamiltonianCycle and HamiltonianCycle is polynomial reducible to VertexCover **Solution:** By definition of NP-Complete

__T__ 8. The following are rules concerning a set T of tasks, $\{T_1, T_2, T_3, T_4, T_5, T_6\}$.

a. T_6 must be done before T_5
b. T_4 must be done before T_2
c. T_1 must be done after T_4 but before T_3
d. T_3 must be done after T_6.

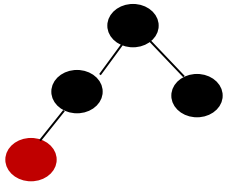
The following is a topological sort of these tasks: $T_4, T_6, T_2, T_5, T_1, T_3$

II. Short Answer [25 points]. Below, 8 problems are given. To get full credit, you must do exactly 5 of these problems. Circle the problem numbers of the problems you are attempting. If you do not circle problem numbers or attempt more than 5 problems, you will be penalized. Each problem is worth 5 points.

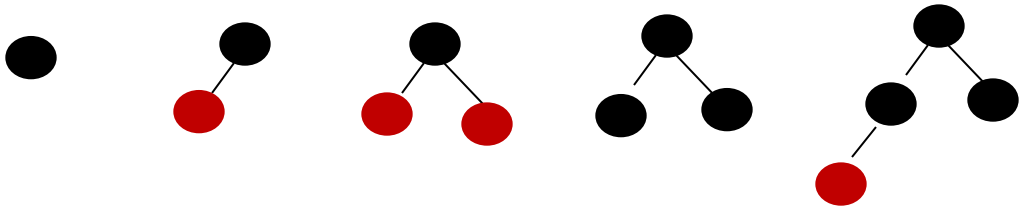
1. Suppose Q is a decision problem and it is known that VertexCover is polynomial reducible to Q. Give an argument to show that Q must be NP-complete.

   Let R be any NP problem. We must show $R \overset{poly}{\dashrightarrow} Q$. But $R \overset{poly}{\dashrightarrow}$ VertexCover (since VertexCover is NP-complete) and, by assumption, VertexCover $\overset{poly}{\rightarrow} Q$. Therefore $R \overset{poly}{\rightarrow} Q$.

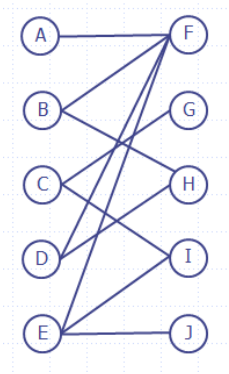
2. Is the following red-black tree derivable? Prove your answer.



   Solution: Yes. The insertion sequence 3,2,4,1 produces precisely this red-black tree.

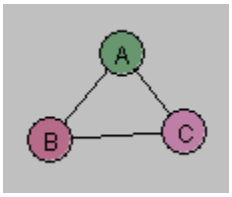


3. Verify Konig's Theorem for the bipartite graph G given below. To do this you will need to find a *maximal matching M* and a *minimal vertex cover U* so that *M* and *U* have the same number of elements.



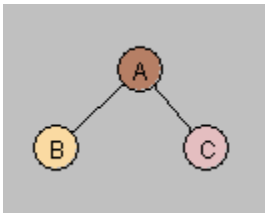
Maximal matching M: BF, CG, DH, EJ [could replace BF with AF; replace EJ with EI]
Minimal vertex cover U: F, C, H, E

4. Is there an undirected graph *G* for which VERTEXCOVERAPPROX produces a vertex cover of minimum possible size? If yes, give an example. If no, explain why not.

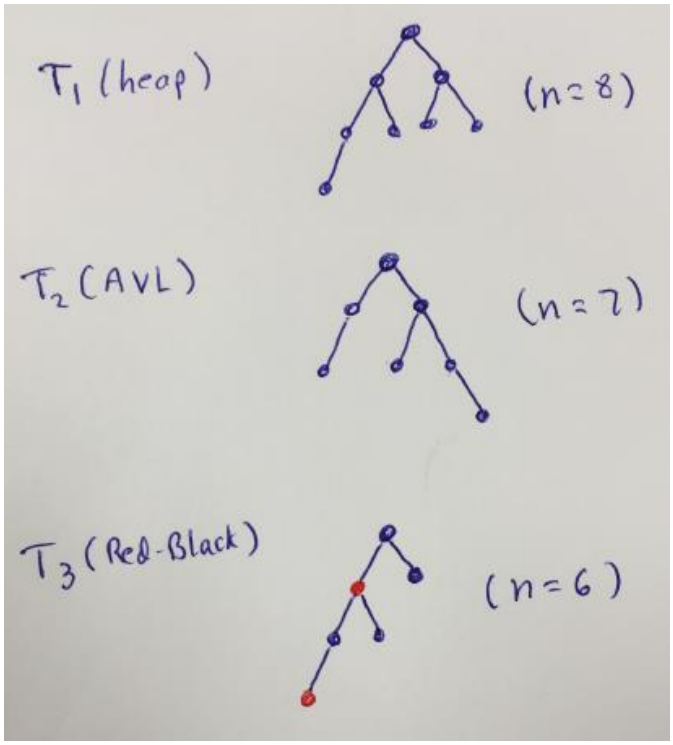


VERTEXCOVERAPPROX produces a cover of size 2, which is the smallest possible cover.

NOTE: The following does not work – true minimal cover has size 1 but VERTEXCOVERAPPROX algorithm produces a cover of size 2:



5. In the space below, draw three binary trees, T_1, T_2, T_3, each with height = 3, with the following characteristics
    a. T_1 is a heap with the smallest possible number of nodes
    b. T_2 is an AVL tree with the smallest possible number of nodes
    c. T_3 is a red-black tree with the smallest possible number of nodes.



6. [*Counts as two problems – circle part a, part b, or both*.] For each of the following, complete the pseudo-code to provide an algorithm meeting the specifications indicated in the Input and Output sections. Then give the asymptotic running time of your algorithm, including an explanation of how you arrived at this running time. NOTE: If your algorithm uses an algorithm that was discussed in class, you do *not* need to reproduce the pseudo-code; instead you can refer to the algorithm by name (for instance, if you will use Breadth First Search in your algorithm, one step of your pseudocode could be "run BFS").

   a. **Algorithm**: IsSimpleCycle(G)
      Input: An undirected graph G.
      Output: TRUE if G itself is a simple cycle, false if not

      Solution
      - Let n = |V|, m = |E|
      - If n ≠ m, return FALSE
      - Pick a vertex v in V.
      - Use a counter to count the number of vertices visited.
      - Perform DFS with starting vertex v but stop the first time the stack needs to be popped.
      - If the last vertex visited is not adjacent to the starting vertex, return FALSE
      - If the number of vertices visited equals n, return TRUE, otherwise FALSE

      Correctness: If G is (can be represented as) a simple cycle, starting with any vertex, DFS will continue until it encounters the starting vertex which will be seen as visited, and so it will try to pop the stack; so the algorithm will stop. The last visited vertex will be adjacent to the starting vertex and number of visited vertices will be n.

      If G is a tree, it will have more vertices than edges, so the algorithm will return FALSE. If G contains a cycle but is not a cycle, then, when the algorithm is run, either the last vertex is not adjacent to the first (if first is not inside a cycle) or the starting vertex is chosen in the middle of a cycle and the algorithm will stop before n vertices have been visited; in either case, the algorithm with return FALSE.

      Running Time of Your Algorithm: O(m+n)

      Explanation: This is a simple variation of DFS.

b. **Algorithm**: IsReachableFrom(G, u, v)
Input: A directed graph G, vertices u, v in G
Output: TRUE if there is a directed path from u to v in G, false otherwise.

Solution
- Perform (directed) DFS starting at u, store each visited vertex in a set V_0, and stop when the stack is empty.
- Return TRUE if v is in V_0, FALSE otherwise.

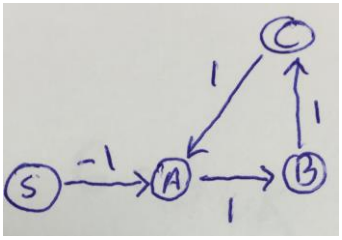
Running Time of Your Algorithm: O(m+n)

Explanation: This is just a simple variation of (directed) DFS

7. Give an example of a *directed* weighted graph *G* and starting vertex *s* having the following four properties (or explain why such an example does not exist)
    a. *G* has at least one edge that has negative weight
    b. All vertices in *G* are reachable from *s*
    c. Dijkstra's Algorithm correctly computes all shortest directed paths from *s* to the vertices in *G*
    d. The dynamic programming algorithm discussed in class for computing shortest paths *does not* produce correct computations of shortest directed paths.

    If you provide an example, you must explain why your graph has all four properties a-d mentioned above.

    Solution:



    a. SA has negative weight
    b. Obvious that all vertices are reachable
    c. There is only one choice at each step of Dijkstra – clearly Dijkstra is correct
    d. The dynamic programming algorithm does not work because there is a directed cycle

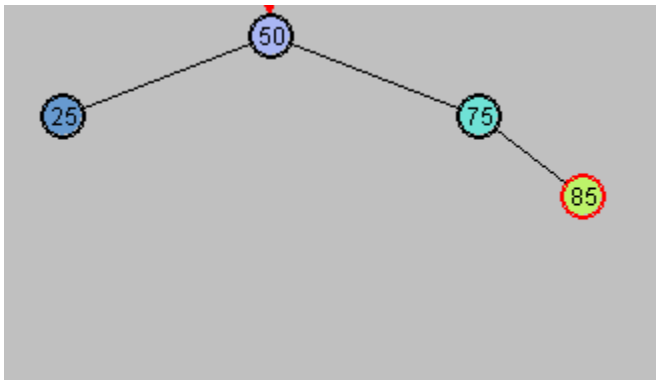
8. Name two P problems for which dynamic programming algorithms are known.

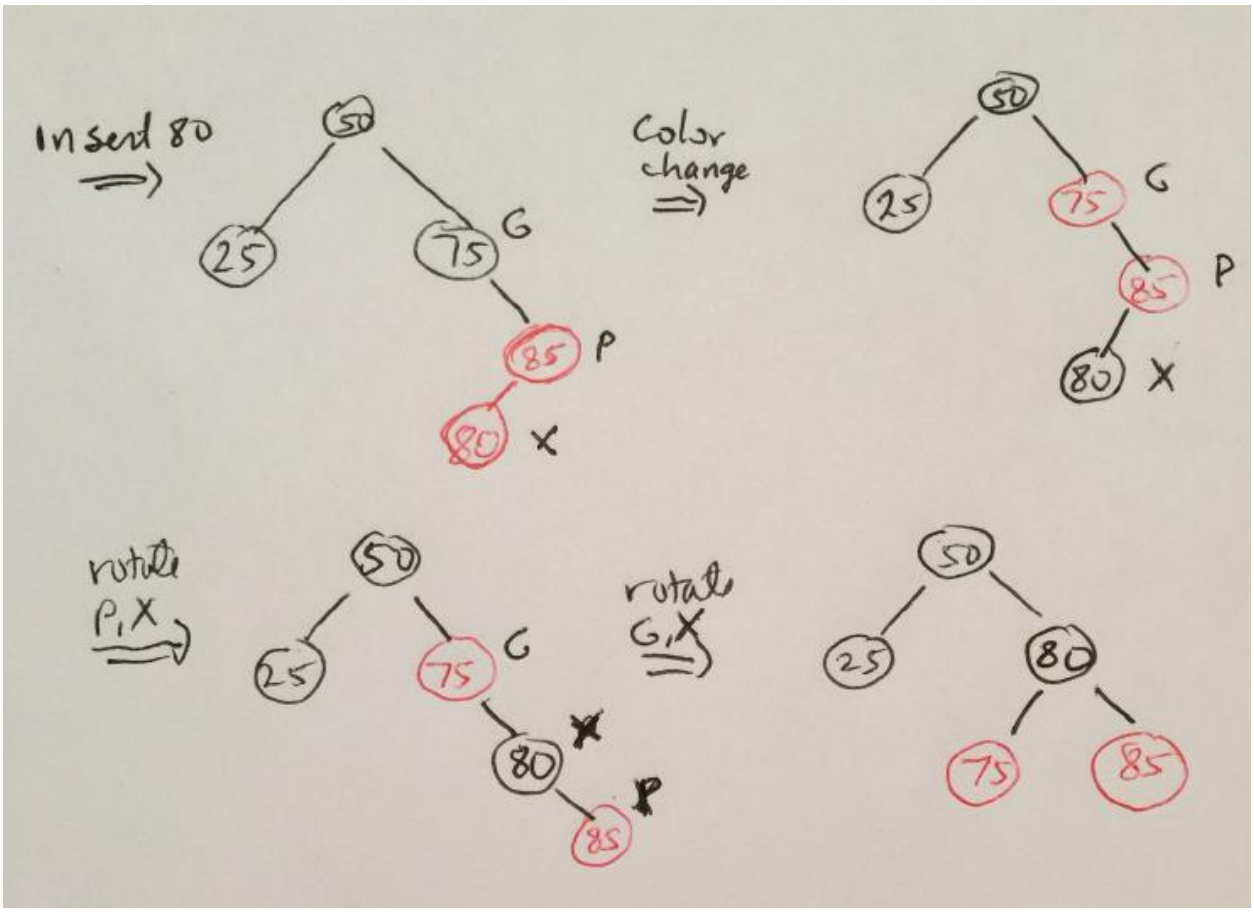
    Solution

    1. Fibonacci
    2. Shortest path

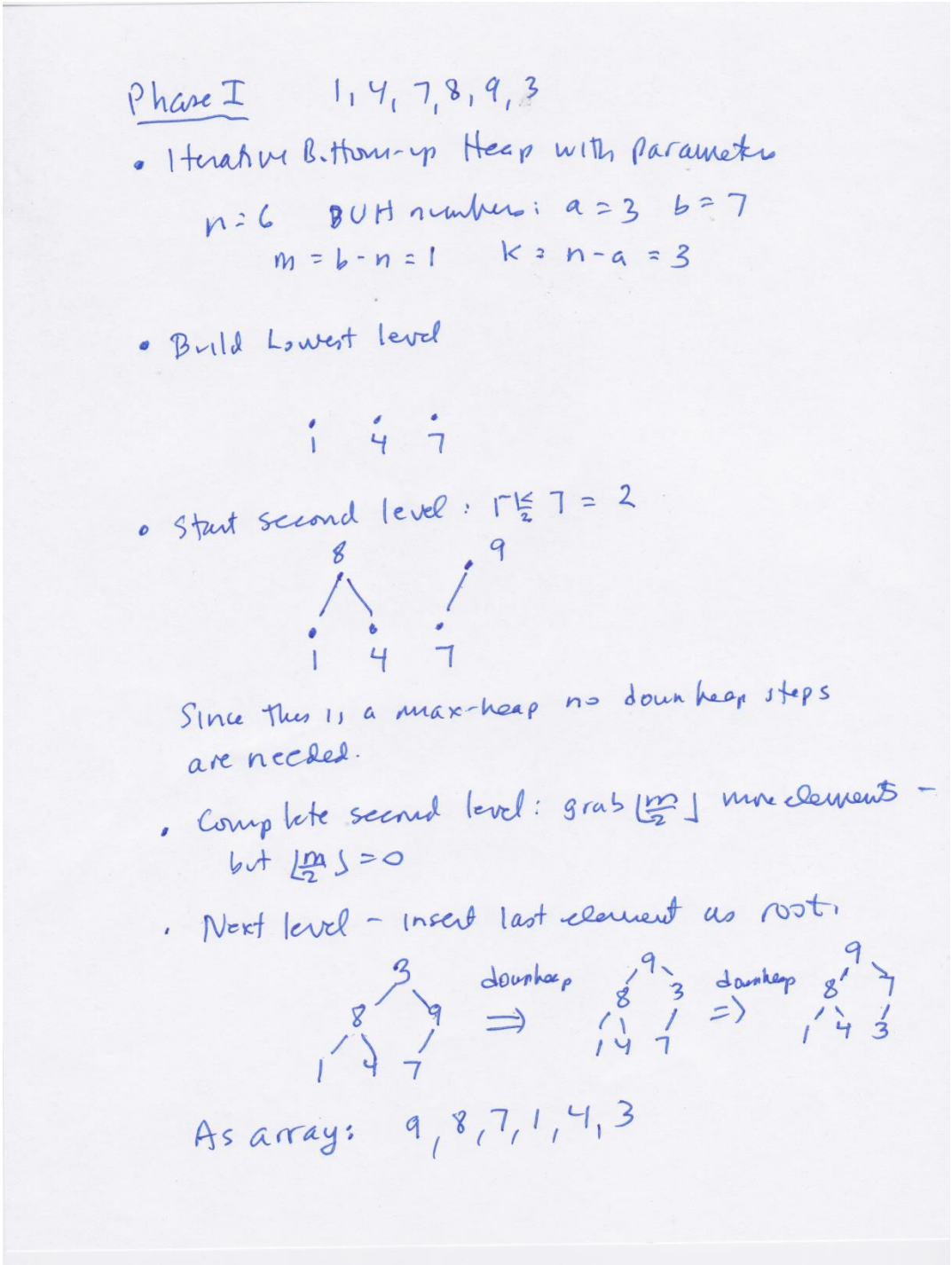
III. Algorithms [42 points]

1. [12 points] *Red-black trees.* Below is a red-black tree. Use the insertion algorithm to insert the value 80 into the tree. To get full credit, you must show all steps.





2. [15 points] *HeapSort.* Use HeapSort to sort the following array: [1, 4, 7, 8, 9, 3]. For Phase I, use the iterative version of Bottom-Up Heap. For Phase II, perform the usual steps to transform an array-based heap into a sorted array. Show all steps and clearly distinguish between Phase I and Phase II.



Phase II 9, 8, 7, 1, 4, 3

remove 9

3 8 7 1 4 | 9

downheap

8 3 7 1 4 | 9

8 4 7 1 3 | 9

remove 8

3 4 7 1 | 8 9

downheap

7 4 3 1 | 8 9

remove 7

1 4 3 | 7 8 9

downheap

4 1 3 | 7 8 9

remove 4

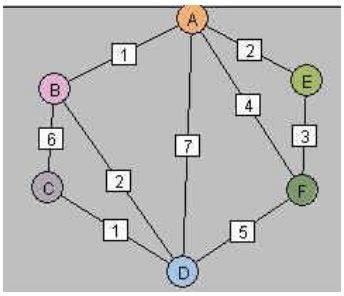
3 1 | 4 7 8 9

remove 3

1 | 3 4 7 8 9

return: [1, 3, 4, 7, 8, 9]

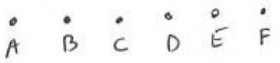
3. [15 points] *Kruskal's Algorithm.* Use Kruskal's algorithm to obtain a minimum spanning tree for the weighted graph given below. To manage clusters, use a tree-based implementation of the DisjointSets data structure. Show how trees evolve after each edge is explored. You must display your final set T of edges, representing a minimal spanning tree for the graph.



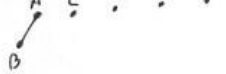
Sorted edges: AB, CD, BD, AE, EF, AF, DF, BC, AD

T = {AB, CD, BD, AE, EF }

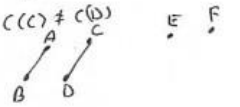
Starting clusters:



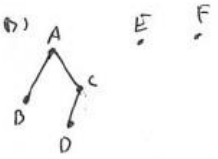
AB C(A) ≠ C(B)



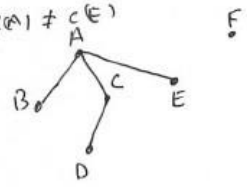
CD C(C) ≠ C(D)



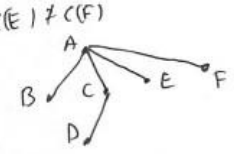
BD C(B) ≠ C(D)



AE C(A) ≠ C(E)



EF C(E) ≠ C(F)



IV. SCI. [3 points] Elaborate upon a parallel between points and topics in Algorithms and one or more SCI principles.