Final

Note

This is a closed-book exam. Each problem accounts for 10 points, unless otherwise marked.

Problems

1. Below is the Mergesort algorithm in pseudocode:

```
Algorithm Mergesort(X, n);
begin  M_Sort(1, n)  end

procedure M_Sort(Left, Right);
begin
    if Right − Left = 1 then
      if X[Left] > X[Right] then swap(X[Left], X[Right])
    else if Left ≠ Right then
            Middle := ⌈½(Left + Right)⌉;
            M_Sort(Left, Middle − 1);
            M_Sort(Middle, Right);
            // the merge part
            i := Left;  j := Middle;  k := 0;
            while (i ≤ Middle − 1) and (j ≤ Right) do
                  k := k + 1;
                  if X[i] ≤ X[j] then
                     TEMP[k] := X[i];  i := i + 1
                  else TEMP[k] := X[j];  j := j + 1;
            if j > Right then
               for t := 0 to Middle − 1 − i do
                   X[Right − t] := X[Middle − 1 − t]
            for t := 0 to k − 1 do
                X[Left + t] := TEMP[1 + t]
end
```

   Given the array below as input, what are the contents of array *TEMP* after the merge part is executed for the first time and what are the contents of *TEMP* when the algorithm terminates? Assume that *TEMP* and X have the same number of entries and each entry of *TEMP* has been initialized to 0 when the algorithm starts.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
15	12	13	3	8	9	14	16	7	1	2	4	11	5	10	6

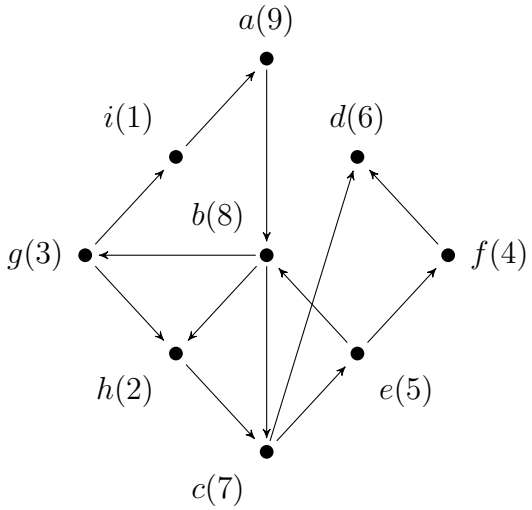
2. Given as input a connected undirected graph G, a spanning tree T of G, and a vertex v, design an algorithm to determine whether T is a valid DFS tree of G rooted at v. In other words, determine whether T can be the output of DFS under some order of the edges starting with v. Please present your algorithm in adequate pseudocode

and make assumptions wherever necessary. Explain why the algorithm is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you get for this problem.

3. Please explain, using an example, why Dijkstra's algorithm does not work for graphs that contain edges with a negative weight.

4. Let $G = (V, E)$ be a connected weighted undirected graph and T be a minimum-cost spanning tree (MCST) of G. Suppose that the cost of one edge $\{u, v\}$ in G is *decreased*; $\{u, v\}$ may or may not belong to T. Design an algorithm either to find a new MCST or to determine that T is still an MCST. Explain why your algorithm is correct and give an analysis of its time complexity. The more efficient your algorithm is, the more points you will be credited for this problem.

5. Run the strongly connected components algorithm on the directed graph shown below, where each vertex is labeled with its ID and DFS number. When traversing the graph, the algorithm should follow the given DFS numbers (from larger to smaller numbers). Show the *High* values as computed by the algorithm in each step.

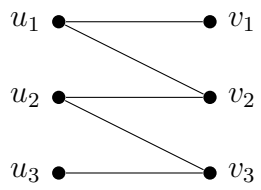


6. Consider designing an algorithm by dynamic programming to determine the length of a longest common subsequence of two strings (sequences of letters). For example, "abbcc" is a longest common subsequence of "abcabcabc" and "aaabbbccc", and so is "abccc".

   (a) Formulate the solution using recurrence relations.

   (b) Present the algorithm in suitable pseudocode, based on the previous recursive formulation. What is the time complexity of your algorithm?

7. The bipartite matching problem (the maximum-cardinality matching problem for bipartite graphs) can be reduced to the network flow problem, which in turn can be reduced to linear programming. Please illustrate the reductions, using the graph below as input to the bipartite matching problem.



(a) Draw the network resulted from the conversion of the bipartite graph.

(b) Give the linear-programming objective function and constraints for the network.

8. To prove that "P = NP" (which seems unlikely though), it suffices to show that some NP-complete problem is in P. Why? Please explain.

9. In the proof (discussed in class) of the NP-hardness of the 3SAT problem by reduction from the SAT problem, we convert an arbitrary boolean expression in CNF (input of the SAT problem) to a boolean expression in 3CNF (where each clause has exactly three literals).

   (a) Please illustrate the conversion by giving the boolean expression that will be obtained from the following boolean expression:

   $$(v + \overline{x}) \cdot (\overline{w} + x + y + z) \cdot (\overline{v} + w + x + \overline{y} + \overline{z}).$$

   (b) The original boolean expression is satisfiable. As a demonstration of why the reduction is correct, please use the resulting boolean expression to show that it is indeed the case.

10. Solve one of the following two problems. (Note: if you try to solve both problems, I will randomly pick one of them to grade.)

    (a) The hitting set problem is as follows.

    > Given a collection C of subsets of a set S and a positive integer k, does S contain a hitting set for C of size k or smaller, that is, a subset $S' \subseteq S$ with $|S'| \leq k$ such that S' contains at least one element from each subset in C?

    Prove that the hitting set problem is NP-complete.

    (b) The subgraph isomorphism problem is as follows.

    > Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, does G_1 have a subgraph that is isomorphic to G_2? (Two graphs are isomorphic if there exists a one-one correspondence between the two sets of vertices of the two graphs that preserves adjacency, i.e., if there is an edge between two vertices of the first graph, then there is also an edge between the two corresponding vertices in the second graph, and vice versa.)

    Prove that the subgraph isomorphism problem is NP-complete.

Appendix

- Below is the algorithm discussed in class for determining the strongly connected components of a directed graph.

```
Algorithm Strongly_Connected_Components(G, n);
begin
    for every vertex v of G do
        v.DFS_Number := 0;
        v.component := 0;
    Current_Component := 0;  DFS_N := n;
    while v.DFS_Number = 0 for some v do
        SCC(v)
end

procedure SCC(v);
begin
    v.DFS_Number := DFS_N;
    DFS_N := DFS_N − 1;
    insert v into Stack;
    v.high := v.DFS_Number;
    for all edges (v, w) do
        if w.DFS_Number = 0 then
            SCC(w);
            v.high := max(v.high, w.high)
        else if w.DFS_Number > v.DFS_Number
                    and w.component = 0 then
                v.high := max(v.high, w.DFS_Number)
    if v.high = v.DFS_Number then
        Current_Component := Current_Component + 1;
        repeat
            remove x from the top of Stack;
            x.component := Current_Component
        until x = v
end
```

- The vertex cover problem: given an undirected graph $G = (V, E)$ and an integer k, determine whether G has a vertex cover containing $\leq k$ vertices. (A *vertex cover* of G is a subset C of vertices such that every edge in G is incident to at least one of the vertices in C.)

  The vertex cover problem is complete.

- The Hamiltonian cycle problem: given an undirected graph G, does G have a Hamiltonian cycle? (A Hamiltonian cycle in a graph is a cycle that contains each vertex, except the starting vertex of the cycle, exactly once.)

  The Hamiltonian cycle problem is NP-complete.