The “Classroom Problems” will be discussed and solved ex tempore at the tutorials in the same week as this problem set is handed out. These are intended to help you in getting started with the material, and will not yield any course credit.

The “Homework Problems” you should solve at home, and attend one of the tutorial sessions in the following week, prepared to present your solutions at class if requested. You earn one tutorial credit point (= 0.33 course points, for a maximum total of 6) for each of these problems that you have solved, and marked as solved at the tutorial session.

The “Demonstration Problems” illustrate further interesting aspects of the material. These will be discussed at the tutorials in the same week as the Homework Problems (time permitting) and printed solutions will be provided. You are encouraged to work on these, but no course credit is given for them.

The “Challenge Problems” are extra problems related to the material, which may be exceptionally difficult and/or require tools beyond the course curriculum.

**Note 1:** Week 43, i.e. 23–27 October, is exam week, and so there are no lectures or tutorials on the course then. Thus, the Classroom Problems from this problem set are discussed at the tutorials in Week 42 and the Homework Problems in Week 44.

**Note 2:** The bonus point deadline for batches P2 and P3 of computerised programming assignments is Sun 29 Oct (cf. MyCourses/Programming assignments). In case you have difficulties in working through the problems, a “walk-in clinic” to assist you is available on Mon 23 Oct, 4–5 p.m., and Fri 27 Oct, 12–1 p.m., in Meeting Room B226 of the CS building.

**Classroom problems:**

1. [Dasgupta et al., Ex. 7.10] For the following network, with edge capacities as shown, find the maximum flow from $S$ to $T$, along with a matching cut:

![Network Diagram]

2. Consider a five-room “house” where every wall has a door, according to the following floorplan:

![Floorplan Diagram]
(a) Is it possible to tour the house by passing through each door exactly once?
(b) Draw a tour that passes through every door except the “front” door in the middle room on the lower row.

Homework problems:

3. [Dasgupta et al., Ex. 5.1/5.2] Consider the following graph.

![Graph Image]

(a) Run Kruskal’s algorithm on this graph. In what order are the edges added to the MST? For each edge in this sequence, give a cut that justifies its addition. (Hint: Note that there could be many justified cuts in each step, you only need to have one.)
(b) Run Prim’s algorithm on the same graph. Whenever there is a choice of vertices, always use alphabetic ordering (e.g. start from vertex A). Draw a table showing the intermediate values of the cost array.
(c) How many minimum spanning trees does the graph have altogether?

4. [Dasgupta et al., Ex. 5.20] Give a linear-time algorithm that takes as input a tree and determines whether it has a perfect matching: a set of edges that touches each vertex exactly once. (Hint: In perfect matching, any vertex of degree one (i.e. leaf) has to match with its neighbor.)

5. [Dasgupta et al., Ex. 5.26] Give an efficient algorithm for the following task. The input is a set of \( n \) variables \( x_1, x_2, \ldots, x_n \) and a set of \( m \) constraints, each of which is either an equality constraint of the form \( x_i = x_j \) or a disequality constraint of the form \( x_i \neq x_j \) for some \( 1 \leq i, j \leq n \). The task is to decide whether the variables can be assigned values so that all the constraints are satisfied. For example, it is not possible to satisfy the four constraints

\[
x_1 = x_2, \ x_2 = x_3, \ x_3 = x_4, \ x_1 \neq x_4.
\]

Your algorithm should have time complexity of at most \( O(m \log n) \). Can you make it \( O(n + m) \)? (Hint: Construct an undirected graph \( G = (V, E) \) such that each vertex represents each variable, and each edge represents each equality constraint. For an inequality constraint, is it ok if the two variables are in the different components? Is it ok if the two variables are in the same component?)
Demonstration problems:

6. [Dasgupta et al., Ex. 7.24] Direct bipartite matching. Let $G = (V_1 \cup V_2, E)$ be a bipartite graph (so that each edge has one endpoint in $V_1$ and one in $V_2$), and let $M \subseteq E$ be a matching in the graph (that is, a set of edges that don’t touch). A vertex is said to be covered by $M$ if it is the endpoint of one of the edges in $M$. An alternating path is a path of odd length that starts and ends with a non-covered vertex, and whose edges alternate between $M$ and $E \setminus M$.

(a) In the bipartite graph below, a matching $M$ is shown in bold. Find an alternating path.

(b) Prove that a matching $M$ is maximum if and only if there does not exist an alternating path with respect to it.

(c) Design an algorithm that finds an alternating path in $O(|V| + |E|)$ time using a variant of breadth-first search.

(d) Give a direct $O(|V| \cdot |E|)$ algorithm for finding a maximum matching in a bipartite graph. 
   (Hint: Note that $\min\{a, b\} \cdot (a + b) \leq 2ab$.)

Solution

(a) DFBG, DFBEAHCI

(b) (If there is an alternating path, then the matching $M$ is not maximum): This direction is easy because we can improve our solution using this alternating path. (If the matching $M$ is not maximum, then there is an alternating path): Let $M^*$ be a maximum matching. Consider the graph $G' = (V, M \cup M^*)$. Notice that, in this graph, each vertex can have degree of at most 2 because each vertex can have degree of at most one in $M$ and in $M^*$. So, $G'$ is a collection of disjoint paths and even cycles. From the fact that $|M| < |M^*|$, there must be a path with odd length that has more edges from $M^*$ than from $M$. Such path is an alternating path.

(c) Observe that any alternating path has odd length, so it must start and end in the different side. First, we pick a free vertex on the left side and run BFS where we only consider a matching edge when we go from right to left. If this BFS finds a free vertex on the right side at some point, then we have an alternating path. If not, we remove all visited vertices, pick the next free vertex on the left side and run BFS again. Keep doing this until we find an alternating path or there is no more free vertex on the left side. (Try to convince yourself why the algorithm is correct, i.e., why we can remove all vertices from previous iterations?)

(d) Since the size of the maximum matching is bounded by $O(\min\{|V|, |E|\})$, we have to find alternating path at most $O(\min\{|V|, |E|\})$ times. This implies the total running time of $O(|V| \cdot |E|)$. 