Exercise 8.1  Branch and bound.

In this exercise, we wish to apply the branch-and-bound method to a classical optimization problem: the so-called Minimum Dominating Set Problem. In this problem, an (undirected) graph \( G = (V, E) \) with \( n = |V| \) nodes and \( m = |E| \) edges is given. We look for a smallest possible set \( D \subseteq V \) of nodes (a minimum dominating set) such that every node in \( V - D \) has a neighbor in \( D \). A node is said to be dominated if it is in \( D \) or has a neighbor in \( D \).

We consider the following example:

In this graph, \( D = \{a, b, c, h, i\} \) is a dominating set from which we cannot eliminate any vertex without losing dominance. However, \( D \) is not as small as possible, so it is not a minimum dominating set. The exercise is to find a minimum dominating set for the above graph by applying the branch and bound method by hand. Proceed as follows:

a) First develop a lower bound on the number of required nodes for a given partial solution. A partial solution is described by two sets \( I \) and \( O \). The set \( I \) contains the nodes that are chosen to be part of the dominating set, while \( O \) contains the nodes that are chosen not to be part of the dominating set.

b) Provide a branching rule to specify which node is the next one to be considered.

c) Perform branch and bound using the answers you gave in a) and b). Draw a complete decision tree and indicate the order of the decisions and the corresponding lower bounds. Identify the final solution.
Exercise 8.2  Algorithm design: hotel booking (part of an exam in August 2009).

A hotel needs a system to handle reservations of its presidential suite. The suite can be booked for any number of nights. Obviously, the suite can be used only by one customer every night. The room can be booked seamlessly, i.e., when a customer leaves on day \( y \), it can be assigned to a new guest from the same day on. The system must support three operations: add bookings (if a reservation is made), delete a booking (if a reservation is canceled), and check whether the suite is available for a given arrival and departure date. A booking \((x, y)\) consists of an arrival date \( x \) and a departure date \( y \), where \( x \) and \( y \) are natural numbers, and \( x < y \).

a) For the first planning step, specify a data structure in which \( n \) bookings can be saved, so that we can check efficiently whether a timespan \((x', y')\) is still free. Briefly describe
   - which data structure you choose,
   - how to create the data structure for \( n \) existing bookings, and how much time is required for the construction,
   - and how the query for a timespan \((x', y')\) can be answered efficiently. Also give the running time to answer each query.

b) Now, the system should be generalized so that the number of bookings is not static anymore. In other words, we want a data structure that efficiently supports the following operations on the (initially empty) set of bookings:
   - **Insert**\((x, y)\): insert a booking \((x, y)\)
   - **Delete**\((x, y)\): delete a booking \((x, y)\)
   - **Query**\((x, y)\): check whether the timespan \((x, y)\) is still available

All the operations should be as efficient as possible, depending on the number of current bookings in the system. Describe your data structure and the individual operations. Provide running times for the operations **Insert**, **Delete** and **Query**.

**Hand-in:** until Wednesday 24th April 2012.