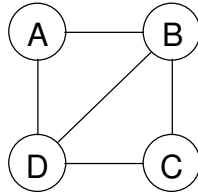


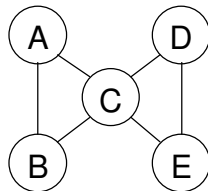
## CS311 Homework #9 Solutions

**10.3.1** The chess decision problem is decidable. A decision tree could be created to analyze all possible moves. In order for the chess position to be deemed a winning position, you need to have at least one move that always leads to a win for every decision that your opponent could possibly make. Given the rules regarding a draw (exact same board position three times in a row, or fifty moves by each player since the last capture or pawn move), you can assume there will always be terminal nodes in this decision tree.

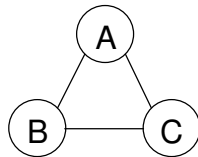
**10.3.3** (a) The following graph has a Hamiltonian circuit (AB-BC-CD-DA), but does not have an Eulerian circuit.



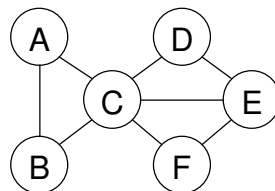
(b) The following graph has an Eulerian circuit (AB-BC-CD-DE-EC-CA), but does not have a Hamiltonian circuit.



(c) The following graph has both an Eulerian and Hamiltonian circuit (AB-BC-CA).



(d) The definition of a cycle from page 32 in the text states, “A cycle is a simple path of a positive length that starts and ends at the same vertex.” Combining the features of part (a) and part (b) of this problem, we can construct a graph that has a cycle including all the vertices (AB-BC-CD-DE-EF-FC-CA) but has neither an Eulerian nor a Hamiltonian circuit.



- 10.3.6** Assume the input  $n$  to our problem is of size  $X$ . Note that in general the size of an object  $z$  is what in Java we would call “ $z.toString().length()$ ”. The loop we need to consider takes  $(n/2 - 1)$  steps. This is about equal to  $2^{X-1}$ . Since this algorithm is not polynomial in its input size, the algorithm is not in class P.
- 10.3.8** This problem asks to reduce the PARTITION problem to an instance of the KNAPSACK problem. Assume we are given an input to the partition problem of a group of numbers  $a_1, a_2, \dots, a_n$ . We need to create an instance of the KNAPSACK problem such that there is a set of  $a$ 's that total to  $\frac{1}{2} \sum a_i$  iff there is a set of items in the knapsack that meet the weight target and the value target. The way to do this is to make  $a_i$  the weight and the value of each item  $i$ . Then the weight target is  $\frac{1}{2} \sum a_i$  and the value target is also  $\frac{1}{2} \sum a_i$ . In order to meet both targets, the sum of the  $i$ 's in the set must be exactly  $\frac{1}{2} \sum a_i$ . Such a set clearly exists iff the answer to the original PARTITION problem is yes.
- 10.3.9** We need to show that CLIQUE, INDEPENDENT-SET (IND-SET), and VERTEX-COVER (VC) are polynomially time reducible to one another.

Start by reducing CLIQUE to IND-SET. Define  $\bar{G}$  to be the graph with the same nodes as  $G$  and an edge from  $(u,v)$  iff  $G$  does not have an edge from  $u$  to  $v$ . The reduction from CLIQUE to IND-SET takes  $(G,k)$  to  $(\bar{G},k)$ . The set of vertices  $X$  is a clique in  $G$  iff  $X$  is a clique in  $\bar{G}$ , so  $G$  has a clique of size  $k$  iff  $\bar{G}$  has an independent set of size  $k$ .

Next, reduce IND-SET to CLIQUE. The same reduction as above does this as well, because  $X$  is an independent set in  $G$  iff  $X$  is a clique in  $\bar{G}$ .

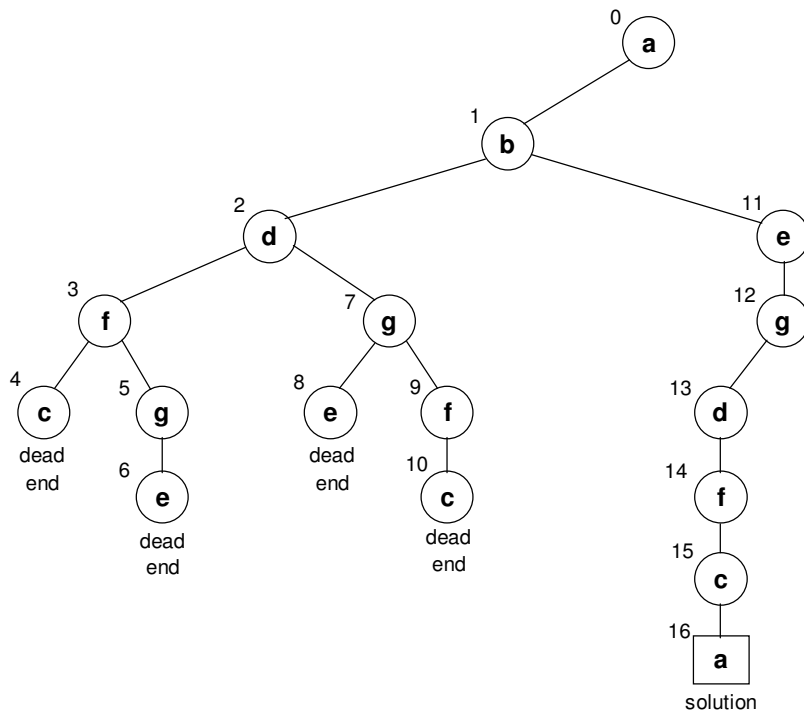
Now reduce IND-SET to VC. This time, we map  $(G,k)$  to  $(G,n-k)$  where  $n$  is the number of vertices in  $G$ . The reason this works is that  $X$  is a vertex cover of  $G$  (every edge of  $G$  touches a vertex of  $X$ ) iff  $\bar{X}$  (the vertices of  $G$  that are not in  $X$ ) is an independent set (no edges of  $G$  are between two vertices of  $\bar{X}$ ). So there exists an independent set  $X$  of size  $k$  iff there exists a vertex cover  $\bar{X}$  of size  $(n-k)$ .

Lastly, we reduce VC to IND-SET. Mapping  $(G,k)$  to  $(G,n-k)$  works in this direction as well, as  $X$  is a vertex cover iff  $\bar{X}$  is an independent set.

These four reductions, alone or in combination, can reduce any of the problems to any of the others. Since we proved in lecture that CLIQUE is NP-complete, all three problems are NP-complete.

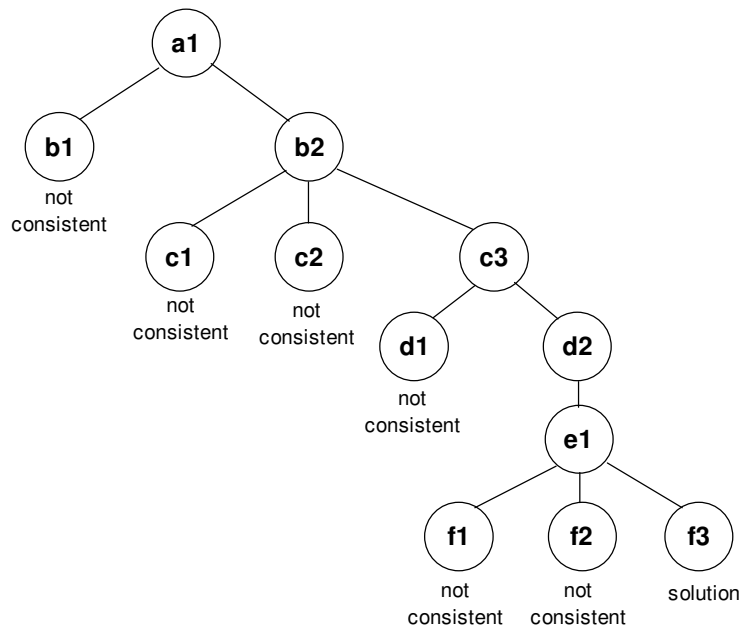
- 10.3.11a** This problem can be modeled as a Hamiltonian circuit. Consider each of the 150 knights as a vertex in a graph. For each knight, put an edge between that knight and all of the other knights he does not quarrel with. Now, if there is a Hamiltonian circuit in this graph, then that gives King Arthur the seating arrangement.

11.1.4 The following tree shows the results of running backtracking on the graph. The algorithm found the Hamiltonian circuit (a-b-e-g-d-f-c-a).



11.1.5 Every time we reach a node, we can try to color it first 1, then 2, and then 3. If the coloring is not consistent with the other nodes, we backtrack and try the other options. We can assume that node a can always be assigned color 1 without loss of generality (as the coloring is arbitrary).

The tree below shows a valid solution to the three coloring problem (a1, b2, c3, d2, e1, f3).



- B9.1**  $H$  has a node  $(u,i)$  for every node  $u$  of  $G$  and every number from 0 through  $(n-1)$ . It has an edge from  $(u,i)$  to  $(u,i+1)$  for every  $u$  and every  $i$ , and also has an edge from  $(u,i)$  to  $(v,i+1)$  for every  $u$  and  $v$  such that  $(u,v)$  is an edge of  $G$ .  $H$  is clearly acyclic. There is a path from  $(s,0)$  to  $(t,n-1)$  in  $H$  iff there is a path from  $s$  to  $t$  in  $G$ .
- B9.2** If  $OPT$  is computable in polynomial time, then we can determine whether  $(x,k)$  is in  $DEC$  very easily. We ask the  $OPT$  algorithm the minimal cost  $c(x,y)$  for any  $y$ . If the answer is less than or equal to  $k$ , then we say yes. Otherwise, we say no.

If  $DEC$  is in  $P$ , we can solve  $OPT$  for a given  $x$  as follows, using the extra assumptions given. First pick any  $y$  and compute  $c(x,y)$ , calling the answer  $z$ . We now know that the right answer is somewhere in the range from 0 to  $z$ . We do a binary search for it, using  $DEC$  to see whether it is in the top or the bottom half of the current range. After at most  $\log z$  calls to  $DEC$ , we will have the answer. The maximum time to run  $DEC$  is polynomial in the length of  $x$ . And since  $z$  is the output of a polynomial time function, it can only have polynomially many bits and its log is also polynomial. The time to compute  $OPT$  is thus the product of two polynomials, which is a polynomial.