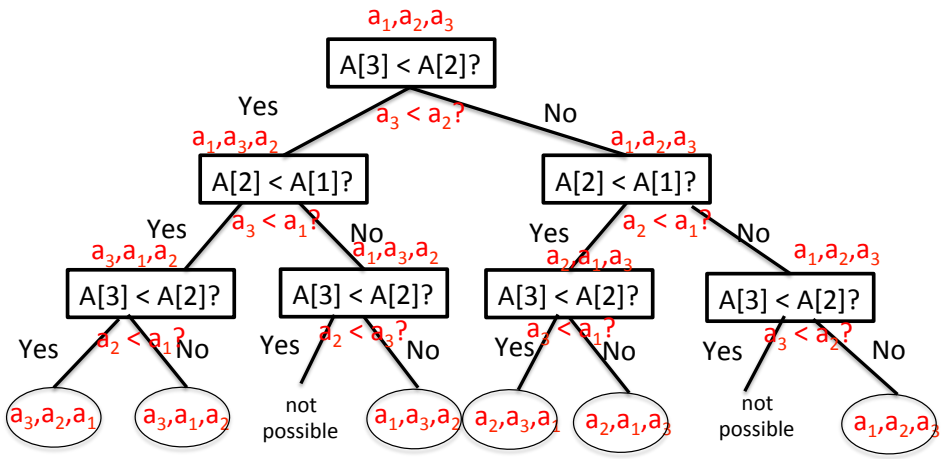
CSCE 411-200, Fall 2016
Homework 6 Solutions

Problem 1: Draw the decision tree for bubble-sort with $n = 3$. Use the code for bubble-sort on page 40 of the textbook.

```
Bubblesort(A)  // A.length = n
  for i = 1 to n - 1
    for j = n downto i + 1
      if A[j] < A[j-1]
        exchange A[j] with A[j-1]
```

Solution: Note that no matter what the input order is, the algorithm will always compare A[3] versus A[2], then A[2] versus A[1], and then A[3] versus A[2] again. The second comparison of A[3] and A[2] serves a purpose when the contents of the entries in A have been rearranged by the exchange (so we are comparing different keys). In the picture, I use A[1], etc. to express the code, and a_1, etc. to indicate the original keys, i.e., originally, A[1] holds a_1, etc.

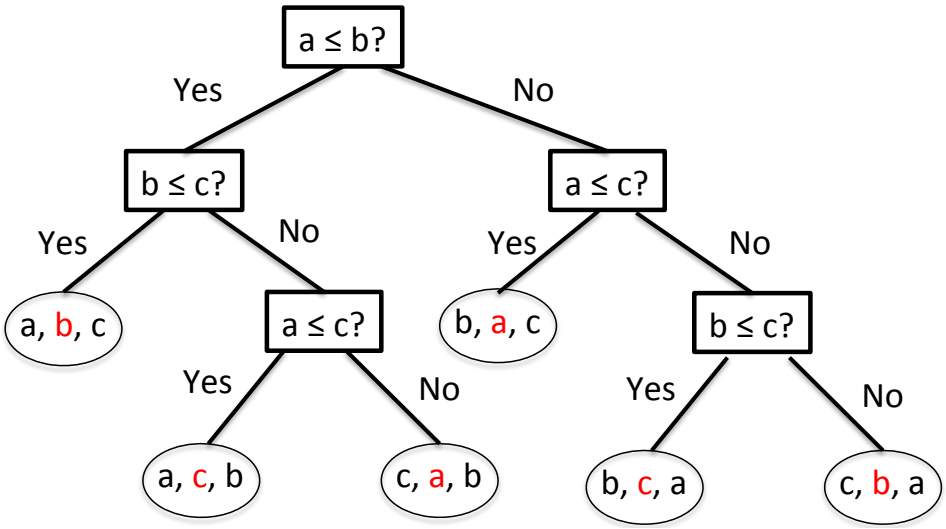


Problem 2: Consider the problem of finding the median of three integers a,b,c.
(a) Describe a comparison-based algorithm for solving this problem.
(b) Draw the decision tree for your algorithm.
(c) What is the worst-case number of comparisons taken by your algorithm?
Solution: (a)

```
int median(int a, int b, int c) {
   if (a <= b) {
      if (b <= c)
         return b;            // a <= b <= c
      else {  // c < b
         if (a <= c)
            return c;         // a <= c < b
         else  // c < a
            return a;         // c < a <= b
      }
   }
   else  { // b < a
```

```
        if (a <= c)
           return a;              // b < a <= c
        else { // c < a
           if (b <= c)
              return c;           // b <= c < a
           else  // c < b
              return b;           // c < b <= a
        }
     }
}
```

(b)



(c) Three (in case $a > b > c$ for instance).

Problem 3: Exercise 34.1-1: Define the optimization problem LONGEST-PATH-LENGTH as the relation that associates each instance of an undirected graph and two vertices with the number of edges in a longest simple path between the two vertices. Define the decision problem LONGEST-PATH = $\{\langle G, u, v, k\rangle : G = (V, E)$ is an undirected graph, $u, v \in V, k \geq 0$ is an integer, and there exists a simple path from u to v in G consisting of at least k edges$\}$. Show that the optimization problem LONGEST-PATH-LENGTH can be solved in polynomial time if and only if LONGEST-PATH $\in P$.

Solution: First, suppose that LONGEST-PATH (the optimization problem) is in P. Then we can solve LONGEST-PATH-LENGTH (the decision problem) in polynomial time like this: For input $\langle G, u, v, k\rangle$, solve the optimization problem to obtain the actual longest path length ℓ; if $\ell \leq k$, then return YES, otherwise return NO.

For the other direction, suppose that LONGEST-PATH-LENGTH (the decision problem) is in P. Then we can solve LONGEST-PATH (the optimization problem) in polynomial time like this: Do binary search on the range $[0, |V|]$ of integers to find the actual longest path between u and v. That is, at first, use the polynomial time algorithm with the bound k equal to $\lceil |V|/2 \rceil$. If the answer is YES, then continue searching in the larger half of the range, otherwise continue searching in the smaller half of the range. There will be $O(\log V)$ calls to the polynomial-time subroutine, so the overall time is polynomial.

Problem 4: Suppose L_1 and L_2 are any two languages in P. (Remember that a language is a set of strings over some finite alphabet; suppose L_1 and L_2 are defined with respect to the same alphabet.) Prove the

following:
(a) The union of L_1 and L_2 is also in P.
(b) The intersection of L_1 and L_2 is also in P.
(c) The complement of L_1 is also in P.
Solution: Since L_1 is in P, there is a polynomial time algorithm A_1 that determines whether or not a string is in L_1; let $p(n)$ be its running time on an input of length n. Since L_2 is in P, there is a polynomial time algorithm A_2 that determines whether or not a string is in L_2; let $q(n)$ be its running time on an input of length n. Below are descriptions of polynomial time algorithms for $L_1 \cup L_2$, $L_1 \cap L_2$, and $\overline{L_1}$.
(a) $L_1 \cup L_2$ consists of every string that is either in L_1 or L_2 (or both).

```
input: x
if A1(x) return true
if A2(x) return true
return false
```

Time is proportional to $p(|x|) + q(|x|)$.
(b) $L_1 \cap L_2$ consists of every string that is in both L_1 and L_2

```
input: x
if A1(x) and A2(x) return true
return false
```

Time is proportional to $p(|x|) + q(|x|)$.
(c) $\overline{L_1}$ consists of every string over L_1's alphabet that is not in L_1.

```
input: x
if A1(x) return false
return true
```

Time is proportional to $p(|x|)$.

Problem 5: Given an undirected graph $G = (V, E)$, an independent set is a subset V' of V such that there is no edge in E connecting any pair of vertices in V'. The independent set decision problem is, given a graph G and an integer k, to determine whether G has an independent set of size at least k. Prove that the independent set decision problem (denoted IS) is NP-complete. For the known NP-complete problem, you may use any one of SAT, 3SAT, HC, TSP, VC, or CLIQUE.
Solution: First show IS is in NP. Let $G = (V, E)$ and k be the input. Given a candidate solution (a subset V' of V), check in polynomial time that there is no edge between any pair of vertices in V'.

Now show CLIQUE $\leq_p$ IS. Let $G = (V, E)$ and k be the input for CLIQUE. Construct input for IS to be G', the complement graph of G, and k. It should be obvious that G' and k can be constructed in time polynomial in the size of G.

Now check that G has a clique of size k if and only if G' has an independent set of size k.

Suppose G has a clique of size k. Then there is a subset V' of V such that there is an edge in G between every pair of vertices in V'. Then in G' there is no edge between any pair of vertices in V', and thus V' is an independent set of G' of size k.

Suppose G' has an independent set of size k. Then there is a subset V' of V such that there is no edge in G' between any pair of vertices in V'. Then in G there is an edge between every pair of vertices in V', and thus V' is a clique of G of size k.

Problem 6: Prove that the 3-COLOR decision problem is NP-complete. Refer to Problem 34-3 for more information about this problem and a description of the reduction to use. You do not need to structure your proof according to parts (d) through (f) if you don't find that helpful. (Skip parts (a) through (c).)

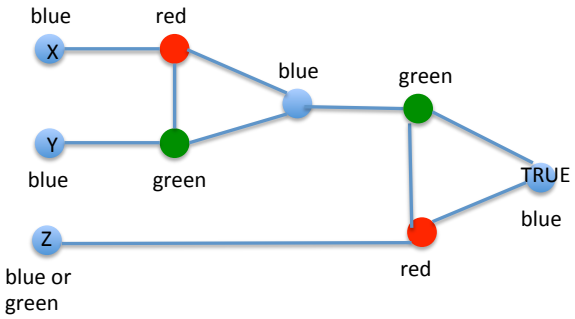
Solution: First show 3-COLOR is in NP. Given a candidate solution (an assignment of colors to vertices), check in polynomial time whether there are at most 3 different colors, and whether no two neighboring vertices are assigned the same color.

Now show that 3SAT is polynomial-time reducible to 3-COLOR, using the reduction described in Problem 34-3. The graph can be constructed in time polynomial in the size of the logical formula, as the total number of vertices is $3 + 2 \cdot n + 5 \cdot m$, where n is the number of variables and m is the number of clauses in the formula, and the number of edges is a constant multiple of the number of vertices.

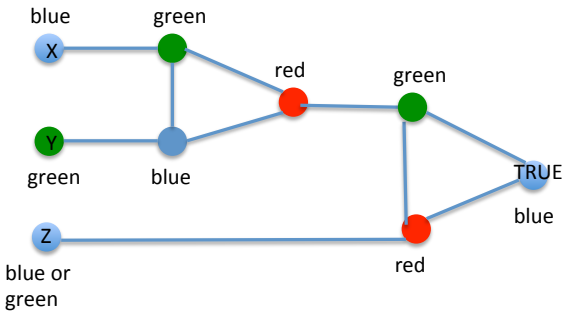
We must show that the formula has a satisfying truth assignment if and only if the graph is 3-colorable.

Assume the formula has a satisfying truth assignment. Color the graph with three colors (red, green, and blue) like this: For each variable x in the formula, if the satisfying truth assignment causes x to be true, then color the vertex for x with blue and the vertex for $\overline{x}$ with green; if it causes x to be false, then vice versa. Color the special vertex TRUE with blue, the special vertex FALSE with green, and the special vertex RED with red. The figures below show how to color the 5-vertex widget corresponding to a clause in the formula. Since the truth assignment is satisfying, at least one of the neighbors on the left (in Fig. 34.20) is blue; recall that TRUE is also blue.

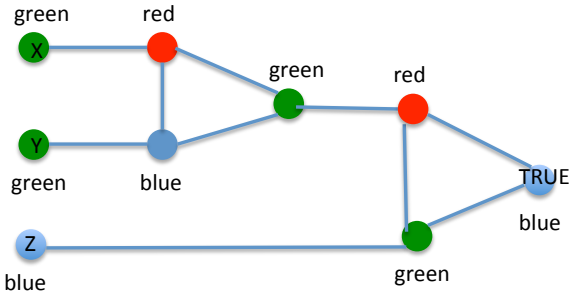
Case 1: Both x and y are blue.



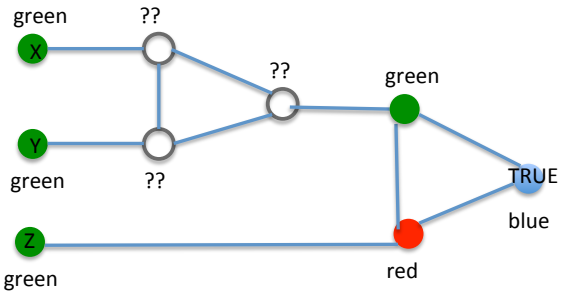
Case 2: Only one of x and y is blue (WLOG, suppose it's x).



Case 3: Neither x nor y is blue, so z must be blue.



Now assume that the graph has a 3-coloring and show the formula has a satisfying truth assignment. Since TRUE, FALSE, and RED form a triangle, each one of these vertices has a different color. WLOG, assume TRUE is colored blue, FALSE is colored green, and RED is colored red. Since each pair of vertices x and $\overline{x}$ form a triangle with RED, either x is blue and $\overline{x}$ is green or vice versa. If x is blue, then assign "true" to the variable x, otherwise assign "false" to x. Show that this truth assignment causes at least one literal in each clause to be true. Suppose for contradiction that there is a clause that has no true literals using this truth assignment. Consider the corresponding widget in the graph, depicted in the figure below.



Since none of the vertices in the top triangle can be green, it isn't possible to properly color the triangle, contradicting the assumption that we were starting with a 3-coloring.

Problem 7: Problem 35-1: **Bin packing.** Suppose we are given n objects, where the size s_i of the i-th object satisfies $0 < s_i < 1$. We wish to pack all the objects into the minimum number of unit-size bins. Each bin can hold any subset of the objects whose total size does not exceed 1.

(a) Prove that the problem of determining the minimum number of bins required is NP-hard. (*Hint:* Reduce from the subset-sum problem.)

The **first-fit** heuristic takes each object in turn and places it into the first bin that can accommodate it. Let $S = \sum_{i=1}^{n} s_i$.

(b) Argue that the optimal number of bins required is at leat $\lceil S \rceil$.

(c) Argue that the first-fit heuristic leaves at most one bin less than half full.

(d) Prove that the number of bins used by the first-fit heuristic is never more than $\lceil 2S \rceil$.

(e) Prove an approximation ratio of 2 for the first-fit heuristic.

(f) Give an efficient implementation of the first-fit heuristic, and analyze its running time.

Solution: (a) Decision-problem version of the bin-packing minimization problem is this: Given a set T of n real numbers, each in $(0, 1]$ (the object sizes), and a bound B (maximum number of bins), is there a way to partition the real numbers into at most B subsets so that each subset sums to at most 1?

To show the bin-packing decision problem is NP-complete, first show it is in NP. A candidate solution would be a partition of T into subsets (i.e., an assignment of each object to a bin). We can verify in polynomial time that at the total number of subsets (bins) is at most B and that the sum of all the numbers assigned to any one subset is at most 1.

Problem says to use subset-sum as the known NP-complete problem, but it's easier to use Partition, which is defined like this: Given a set A of positive integers, does there exist a subset A' of A such that the sum of all the elements in A' is equal to the sum of all the elements in $A - A'$?

Now show Partition is polynomial-time reducible to bin-packing. Let $A = \{a_1, \ldots, a_n\}$ be a Partition input. Let Z be the sum of all the elements of A. Convert A into the bin-packing input T and B, where $T = \{t_1, \ldots t_n\}$, where each $t_i = a_i/(Z/2)$, and $B = 2$.

T and B can be computed in polynomial time, as we just have to do $O(n)$ additions to compute Z and $O(n)$ divisions to compute T.

Now show the if-and-only-if property.

Suppose A has a partition, i.e., there exists a subset A' of A such that the sum of all elements in A' equals the sum of all elements in $A - A'$, which is $Z/2$. The elements of T corresponding to the elements of A' sum up to $(Z/2)/(Z/2) = 1$. Thus they can be stored in one bin. Similarly, the elements of T corresponding to the elements of $A - A'$ can be stored in a second bin. Thus the elements of T can be stored in $B = 2$ bins.

Now supose the elements of T can be stored in $B = 2$ bins. First note that the sum of all the elements of T is $(2/Z) \cdot Z = 2$. Thus both of the two bins are completely full. Let T' be the subset of T stored in the first bin and let A' be the set of elements of A that correspond to the elements in T'. Sinc ethe elements in T' add up to 1, the elements in T' add up to $Z/2$. Similarly, the elements in A that correspond to the elements in the second bin add up to $Z/2$. So A has a partition.

(b) The optimal number of bins is at least $\lceil S \rceil$, since the exact amount of space needed is S, but the bins come in unit sizes so we have to round up.

(c) First-fit leaves at most one bin less than half full, since otherwise, if there were two bins less than half full, the algorithm would have filled up the first bin before even starting to use the second bin.

(d) The number of bins used by first-fit is at most $\lceil 2S \rceil$, from part (c).

(e) The approximation ratio for first-fit is $\lceil 2S \rceil / \lceil S \rceil$, which is at most 2, by parts (d) and (b). (Recail that $\lceil 2S \rceil \leq 2\lceil S \rceil$.)

(f) First fit algorithm: Main data structure is an array `Bins[1..n]` of structs with two components, `Bins[i].objects` (set of object ids, initially empty) and `Bins[i].capacity` (how much space remains, initially 1).

```
input: S[1..n]   // sizes of the n objects
for i := 1 to n do
   j := 1
   while (Bins[j].capacity < S[i]) j++
   insert i into Bins[j].objects
   subtract S[i] from Bins[j].capacity
```

Time is $O(n^2)$ since we need to search through at most n bins for each of the n objects.