1. Let Φ be a boolean formula in conjunctive normal form, with exactly three literals in each clause. Recall that an assignment of boolean values to the variables in Φ ***satisfies*** a clause if at least one of its literals is TRUE. The ***maximum satisfiability problem*** for 3CNF formulas, usually called MAX3SAT, asks for the maximum number of clauses that can be simultaneously satisfied by a single assignment.

   Solving MAX3SAT exactly is clearly also NP-hard; this question asks about approximation algorithms. Let *Max3Sat*(Φ) denote the maximum number of clauses in Φ that can be simultaneously satisfied by one variable assignment.

   (a) Suppose we assign variables in Φ to be True or False using independent fair coin flips. Prove that the expected number of satisfied clauses is at least $\frac{7}{8}$*Max3Sat*(Φ).

   Solution: Each clause contains three distinct literals. If two of those literals are a variable x and its negation $\overline{x}$, then the clause is satisfied by every assignment. Otherwise, the literals come from three distinct variables, whose values are chosen independently and uniformly at random. Thus, each clause of Φ is satisfied with probability *at least* $7/8$. Linearly of expectation now implies that the expected number of satisfied clauses is *at least* $7/8$ of the total number of clauses. The total number of clauses cannot be smaller than *Max3Sat*(Φ). ■

   (b) Let k^+ denote the number of clauses satisfied by setting every variable in Φ to TRUE, and let k^- denote the number of clauses satisfied by setting every variable in Φ to FALSE. Prove that $\max\{k^+, k^-\} \geq$ *Max3Sat*$(\Phi)/2$.

   Solution: A clause is satisfied by setting every variable to TRUE if and only if it contains a positive literal, meaning a variable that is not negated. Thus, k^+ is the number of clauses containing positive literals.

   Similarly, a clause is satisfied by setting every variable to FALSE if and only if it contains a negative literal, meaning a negated variable. Thus, k^- is the number of clauses containing negative literals.

   Every clause contains either a positive literal or a negative literal, so $k^+ + k^-$ is greater than or equal to the total number of clauses. It follows that $\max\{k^+, k^-\}$ is at least half the total number of clauses. The total number of clauses cannot be smaller than *Max3Sat*(Φ). ■

   (c) Let *Min3Unsat*(Φ) denote the *minimum* number of clauses that can be simultaneously left *unsatisfied* by a single assignment. Prove that it is NP-hard to approximate *Min3Unsat*(Φ) within a factor of $10^{10^{100}}$.

   Solution: Suppose there is a polynomial-time $10^{10^{100}}$-approximation algorithm for MIN3UNSAT. Let Φ be an arbitrary 3CNF formula. If Φ is satisfiable, then *Min3Unsat*$(\Phi) = 0$, so the approximation algorithm must output 0. On the other hand, if Φ is not satisfiable, then *Min3Unsat*$(\Phi) > 0$, so the output of the approximation algorithm must be at least *Min3Unsat*$(\Phi)/10^{10^{100}} > 0$. Thus, Φ is satisfiable if and only if our approximation algorithm returns zero. In other words, we've just described a polynomial-time algorithm for 3SAT! ■

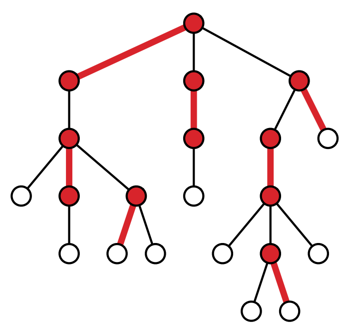
2. Consider the following algorithm for approximating the minimum vertex cover of a connected graph G: ***Return the set of all non-leaf nodes of an arbitrary depth-first spanning tree***. (Recall that a depth-first spanning tree is a rooted tree; the root is not considered a leaf, even if it has only one neighbor in the tree.)

   (a) Prove that this algorithm returns a vertex cover of G.

   Solution: Let T be a depth-first spanning tree of G, and let N be the set of nodes in G that are not leaves in T. A vertex v is a leaf in T if and only if the depth-first search visits every neighbor of v before it visits v. Thus, two leaves in T cannot be adjacent in G. Conversely, for any edge in G, at least one of its endpoints must be in N. We conclude that N is a vertex cover of G. ■

   (b) Prove that this algorithm returns a 2-approximation to the smallest vertex cover of G.

   Solution: Let T be a depth-first spanning tree of G. We construct a *matching* M in T—a subgraph of T in which every node has degree 1—using the following recursive algorithm. Perform a preorder traversal of T, visiting the children of each node in arbitrary order; initially M is empty and every node is unmarked. Whenever the traversal visits an unmarked node v whose parent u is also unmarked, add edge uv to M and mark both u and v.[1]



The number of edges in the matching is at least half the number of non-leaf nodes in the tree.

   By construction, every non-leaf node in T is marked and therefore adjacent to exactly one edge in M. (Some leaves are also marked, but that only helps us.) Thus, if T has k non-leaf nodes, then M contains at least $k/2$ edges. Every vertex cover contains at least one node incident to each edge in M; thus, the minimum vertex cover has at least $k/2$ nodes. We conclude that our algorithm computes a vertex cover that is at most twice as large as the minimum vertex cover. ■

   (c) Describe an infinite family of connected graphs for which this algorithm returns a vertex cover of size *exactly* $2 \cdot \textsc{Opt}$. This family implies that the analysis in part (b) is tight. *[Hint: First find just **one** such graph, with few vertices.]*

   Solution: Suppose the input graph G is a path of length $2k$ with vertices $v_0, v_1, \ldots, v_{2k}$. If we start our depth-first search at v_0, then v_{2k} is the only leaf in the depth-first spanning tree, so our algorithm returns a vertex cover of size $2k$. On the other hand, the odd-index vertices $v_1, v_3, \ldots, v_{2k-1}$ define a vertex cover of G of size k. ■

[1]This is just an implementation of the "dumb" approximation algorithm described in class, where we use DFS to discover unmarked edges.

3. Consider the following modification of the "dumb" 2-approximation algorithm for minimum vertex cover that we saw in class. The only change is that we return a set of edges instead of a set of vertices.

```
APPROXMINMAXMATCHING(G):
  M ← ∅
  while G has at least one edge
      uv ← any edge in G
      G ← G \ {u, v}
      M ← M ∪ {uv}
  return M
```

(a) Prove that the output subgraph M is a *matching*—no pair of edges in M share a common vertex.

Solution: Let uv and vw be arbitrary edges in G that share a common vertex v. If the algorithm ever adds uv to M, it also immediately removes vw from G, so vw cannot be added to M later. Conversely, if the algorithm ever adds vw to M, it cannot add uv to M later. Thus, at most one of those two edges appears in M. ■

(b) Prove that M is a *maximal* matching—M is not a proper subgraph of another matching in G.

Solution: Every edge that is removed from G either belongs to M or shares a vertex with an edge in M. The algorithm terminates when all edges are removed. Thus, every edge that in not in M shares a vertex with at least one edge in M. ■

(c) Prove that M contains at most twice as many edges as the *smallest* maximal matching in G.

Solution: For any two maximal matchings M and M', each edge in M is incident to at most two edges in M', and therefore $|M| \le 2|M'|$. Let M' be the smallest maximal matching. ■

Solution: Every edge in M is incident to at least one matched vertex in any other maximal matching M'. Thus, the number of edges is M at most the number of vertices in M', which is twice the number of edges in M'. Let M' be the smallest maximal matching. ■

Solution: Let *OPTMM* denote the number of edges in the smallest maximal matching, let *OPTVC* denote the number of vertices in the smallest vertex cover.

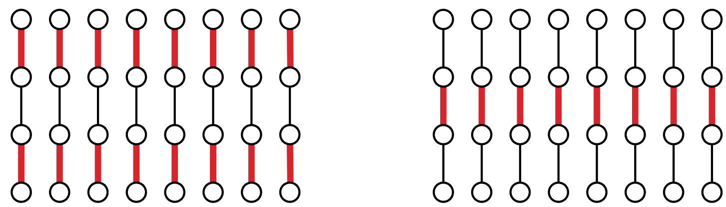
The vertices of *any* maximal matching M' are a vertex cover for G. (If there is an uncovered edge, then M' is not maximal after all.) In particular, the vertices in the *smallest* maximal matching are a vertex cover for G. Thus, $2 \cdot OPTMM \ge OPTVC$.

We proved in class that the vertices of M are a 2-approximation of the smallest vertex cover. Thus, $2 \cdot |M| \le 2 \cdot OPTVC$.

Combining these two inequalities gives us $|M| \le OPTVC \le 2 \cdot OPTMM$. ■

(d) Describe an infinite family of graphs G such that the matching returned by APPROXMINMAX-MATCHING(G) contains exactly twice as many edges as the smallest maximum matching in G. This family implies that the analysis in part (c) is tight. *[Hint: First find just* ***one*** *such graph, with few vertices.]*

Solution: Let G be the disjoint union of k paths of length 3; the smallest maximal matching in G consists of the middle edge of each path and therefore has size k. If ApproxMinMaxMatching(G) checks the first and last edge of each path before any of the middle edges. Then ApproxMinMaxMatching(G) returns a matching of size $2k$ consisting of the first and last edge of each path.



The matching returned by ApproxMinMaxMatching(G) and the smallest maximal matching.

■