Design and Implementation of Algorithms
Solution Sketch for Homework 8

- Exercise 3 of Chapter 12

  (a) A DNF formula ϕ is satisfiable if and only if at least one of its terms is satisfiable. So our algorithm will simply check each term independently to see if it is satisfiable. If none of the terms are satisfiable, we declare that ϕ is not satisfiable; if even one of the terms is satisfiable, we declare that ϕ is satisfiable.

  To determine whether a term is satisfiable, we check whether it contains in its literals both a variable and its complement. If it does, the term is not satisfiable, otherwise it is satisfiable.

  (b) The error is that the algorithm to convert CNF to DNF is not a polynomial time algorithm. Thus, doing the conversion, and then applying a polynomial time algorithm for DNF, does not result in a polynomial time algorithm for 3SAT. Why is the conversion algorithm not polynomial? Consider a 3SAT formula with m clauses. The corresponding DNF expression can have 3^m terms in the worst case, which is exponentially long.

- Exercise 33 of Chapter 12

  We show that the problem is NP-hard via a polynomial time reduction from 3SAT. Our reduction algorithm takes as input a 3CNF formula ϕ. Suppose ϕ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$. The reduction algorithm creates an instance of solitaire with an $m \times n$ grid. Thus, row i of the grid corresponds to clause C_i, and column j of the grid corresponds to variable x_j. We fill the grid as follows, for each $1 \leq i \leq m$ and $1 \leq j \leq n$: if C_i contains x_j as a literal, place a blue stone in location (i, j); if C_i contains $\bar{x_j}$ as a lietral, place a red stone in location (i, j); if C_i contains neither x_j nor $\bar{x_j}$ as a literal, then leave location (i, j) empty.

  The reduction algorithm then checks if the constructed instance of Solitaire is solvable, by invoking a "magical" subroutine for Solitaire. If the subroutine outputs "yes", our algorithm declares that ϕ is satisfiable; if the subroutine outputs "no", then our algorithm declares that ϕ is not satisfiable.

  Ideally, I should give an explanation here for why the reduction algorithm is correct. I am hoping you can convince yourself of the correctness. I will skip such an explanation for the next two problems as well.

- Exercise 34 (a) of Chapter 12

  We show that the problem is NP-hard via a polynomial time reduction from minimum vertex cover. Our reduction algorithm takes as input an instance of minimum vertex cover, an undirected graph $G = (V, E)$. The reduction algorithm constructs a $|E| \times |V|$ grid. Thus, rows corresponds to edges and columns correspond to vertices. For each edge $e = (u, v) \in E$, we place a stone in locations (e, u) and (e, v) of the grid. Thus,

we have constructed an instance of the game. (Note that each row has exactly two stones on it; at least one of the corresponding columns has to be cleared in a valid solution to the game. This captures the requirement that for a particular edge, at least one of the two incident vertices has to be picked in a vertex cover.)

Having constructed the game instance, the reduction algorithm invokes the "magical" subroutine for solving it to find the minimum number of columns that need to be cleared. It outputs that number as the size of the minimum vertex cover in G.

- Exercise 34 (b) of Chapter 12

  We show that the problem is NP-hard via a polynomial time reduction from maximum independent set. Our reduction algorithm takes as input an instance of maximum independent set, an undirected graph $G = (V, E)$. It constructs a game instance using *exactly* the same algorithm used in 34 (a).

  Having constructed the game instance, the reduction algorithm invokes the "magical" subroutine for solving it to find the maximum number of columns that can be cleared. It outputs that number as the size of the maximum independent set in G.