This assignment is designed to be an introduction to the concepts used in our study of NP-completeness later in the semester. Some of the questions are trivial; some are subtle. You should have enough background to answer these questions.

1 Comparison Networks: An Analogy

See the Wikipedia article on *Sorting Networks*. Just read down to, but not including, the zero-one principle. We will be concerned with comparison networks, in general, not just sorting networks.

The *size* of a comparison network is the number of comparators, and the *depth* is the number of levels of comparators. For example, in the Wikipedia article, the initial sorting network for four inputs has size five and depth three. The parallel bubble sort network for six inputs has size fifteen and depth nine. In general, a parallel bubble sort network for n inputs has size $n(n-1)/2$ and depth $2n-3$.

A minmax network inputs a list of size n and outputs a list of size n with the smallest value at the beginning of the list the largest value at the end of the list. For example if the input is the list of size $n = 8$

$$(40, 80, 30, 60, 10, 70, 20, 50).$$

The output would be

$$(10, ?, ?, ?, ?, ?, ?, 80).$$

1. Let n be a power of 2.
   (a) Show how to construct an efficient minmax network with n inputs. Primarily minimize the depth and secondarily minimize the size. Just describe the network; do not justify.
   (b) What is the (exact) size of your network?
   (c) What is the (exact) depth of your network?

It turns out that comparison networks are very important on Mars, where they have many applications. A *merging network* inputs two sorted lists each of size $n/2$ and outputs the total list of values in order. The Martians have discovered merging networks with depth about $\lg n$. (Such merging networks were independently discovered on Earth by Batcher.)

The Martians are especially interested in sorting networks. They have proven that any sorting network must have depth at least $\lg n$ (and size at least about $n \lg n$). They know that there are sorting networks with depth $O((\log n)^2)$, but do not know whether they can attain depth $O(\log n)$.

A problem similar to sorting that Martians often need to solve is *half sorting*: The input is two lists each of size $n/2$, and the output is the two lists independently sorted. For example if the input is the list of size $n = 8$

$$(40, 80, 30, 60, 10, 70, 20, 50).$$

A standard sorting network would output

$$(10, 20, 30, 40, 50, 60, 70, 80).$$

A half sorting network would output

$$(30, 40, 60, 80, 10, 20, 50, 70).$$

Not surprisingly, sorting and half sorting are closely related.

2. (a) Show that if sorting can be solved on a comparison network in depth $O(\log n)$ then half sorting can be solved on a comparison network in depth $O(\log n)$.

   (b) Show that if half sorting can be solved on a comparison network in depth $O(\log n)$ then sorting can be solved on a comparison network in depth $O(\log n)$.[1]

Your two "reductions" show:

Corollary. There exists a depth $O(\log n)$ sorting network if and only if there exists a depth $O(\log n)$ half sorting network.

The Martians extended this result to show that many other important comparison problems are equivalent to sorting and half sorting: Any one of these special problems is solvable in depth $O(\log n)$ if and only if all of them are solvable in depth $O(\log n)$.

Recently two Martian computer scientists Koco and Nevil made a startling discovery.

Definition. A function $f(n)$ is *polylog(n)* if there exists a constant k such that $f(n) = O((\log n)^k)$. We will just say *polylog* when the parameter n is implicit.

Theorem. If there exists a depth $O(\log n)$ sorting network then, for every problem solvable on a comparison network in polylog depth there exists a depth $O(\log n)$ comparison network.

For details, see their article in the prestigious *Martian Online Journal Of Computer Science* (otherwise known as *MOJO CS*).

To finish the analogy: Let **L** (for **L**og depth) be the class of problems solvable in depth $O(\log n)$ on a comparison network with n inputs. Let **PL** (for **P**oly**L**og depth) be the class of problems solvable on a comparison network in polylog depth. The theorem of Koco and Nevil shows that sorting is **PL**-complete (in other words, if sorting is in **L**, then every problem in **PL** is also in **L**). The open problem on Mars is: Does **L** = **PL**?

3. Is half sorting **PL**-complete? Justify.

Note that on Earth, **L** and **PL** are used to represent different classes. Also on Earth, we have discovered an $O(\log N)$ depth sorting network, but the constants are too large to be practical.

[1] A plausible solution would be to take a half sorting network with $2n$ inputs and only use the top n inputs and outputs. This is a way that a half sorting network could be used to sort, but it is not a sorting network with n inputs.

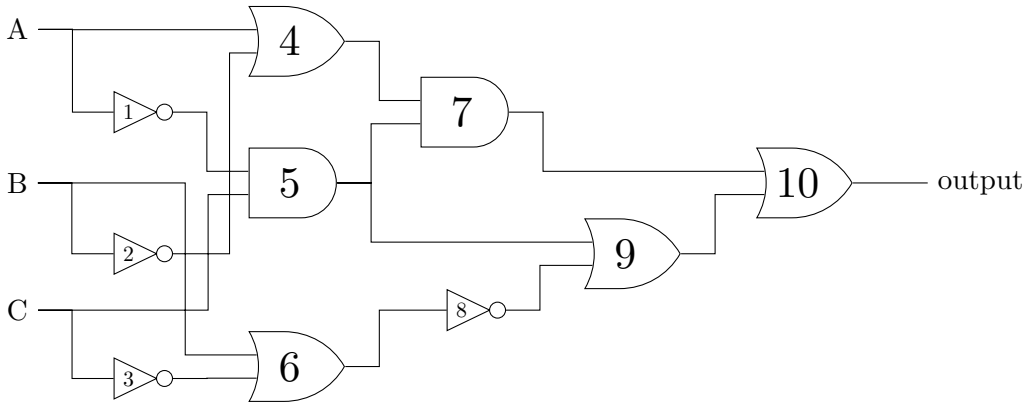
2 Boolean Formula Evaluation

4. (a) Consider the formula

$$((A \wedge \overline{B}) \vee (\overline{A} \wedge C)) \wedge (B \wedge \overline{C})$$

with assignment to the variables $A, B \equiv$ TRUE, $C \equiv$ FALSE. Evaluate the formula. You do not need to show your work.

(b) Consider the Boolean circuit, with assignment to the inputs $A, B \equiv$ TRUE, $C \equiv$ FALSE. Evaluate the Boolean circuit. Show your work by indicating the truth value produced by



each gate. Use the table in the back of this assignment.

The senario: You are working as a computer programmer for the Alpha Beta Gamma Software company. It is your dream job.

Your manager calls you into the office with the following comment:

> We hired you because you have a Computer Science degree from one of those fancy colleges. Nobody else here at Alpha Beta Gamma Software has your education or ability. We are moving into the business of Boolean formula evaluation. Starting next month, every morning we will be receiving a large number of large Boolean formulas. For each formula, the assignment of TRUE's and FALSE's to the variables will be given. We will evaluate each formula. We at Alpha Beta Gamma believe that we can rely on you to write a lightning fast program to evaluate these Boolean formulas.

You have no idea how to write such a program. You scour the internet but cannot find a satisfactory program to evaluate Boolean formulas. However, you do find a great program to evaluate Boolean circuits.

5. It turns out that it is easy to evaluate Boolean formulas directly in linear time. The spirit of this problem is to avoid doing so!

(a) Assume that you are allowed to use the program that evaluates Boolean circuits *only once*. Briefly explain how you would use the Boolean circuit evaluation program to evaluate a Boolean formula.

(b) Show what you would do on the following formula (from Part 4a):

$$((A \wedge \overline{B}) \vee (\overline{A} \wedge C)) \wedge (B \wedge \overline{C})$$

6. Assume the scenario is reversed: You need to evaluate Boolean circuits, but you have a program that evaluates Boolean formulas. Analogously to the previous problem, it is easy to evaluate Boolean circuits directly in linear time. The spirit of this problem is to avoid doing so!

   (a) Assume that you are allowed to use the program that evaluates Boolean formulas *only once*. Briefly explain how you would use the Boolean formula evaluation program to evaluate a Boolean circuit.

   (b) Show what you would do on the circuit in Part 4b.

3 Boolean Formula Satisfiability

A Boolean formula is *satisfiable* if there is an assignment of TRUE's and FALSE's to the variables that makes the formula TRUE. A Boolean circuit with one output wire is *satisfiable* if there is an assignment of TRUE's and FALSE's (or 1's and 0's) to the inputs that makes the output wire TRUE (or 1).

7. This question works with Boolean formulas in Conjuntive Normal Form (CNF). If you are not sure what that is, look it up. A CNF formula is in *k-CNF* if every clause has exactly k literals (for some natural number k).
   (a) Give a 2-CNF formula that is satisfiable, where no variable occurs twice in the same clause. No justification needed.
   (b) Give a 2-CNF formula that is not satisfiable, where no variable occurs twice in the same clause. No justification needed.
   (c) Consider the following 3-CNF formula (with four variables and sixteen clauses):

$$(A \vee B \vee C)(A \vee B \vee \overline{C})(A \vee \overline{B} \vee C)(\overline{A} \vee B \vee \overline{C})(A \vee B \vee \overline{D})(\overline{A} \vee B \vee \overline{D})(A \vee \overline{B} \vee \overline{D})(\overline{A} \vee B \vee D)$$
$$(A \vee C \vee D)(A \vee C \vee \overline{D})(\overline{A} \vee C \vee D)(\overline{A} \vee \overline{C} \vee D)(B \vee C \vee D)(B \vee \overline{C} \vee D)(\overline{B} \vee C \vee D)(\overline{B} \vee \overline{C} \vee \overline{D})$$

   After much struggling, you discover that the formula is satisfiable with the assignment $A, D \equiv$ FALSE and $B, C \equiv$ TRUE. Confirm this by circling exactly one literal in each clause such that this assignment makes the clause TRUE. Use the table in the back of this assignment.

8. Imagine that there is a large Boolean formula (not necessarily in CNF) written on the whiteboard, where n is the number of connectives (AND's, OR's, and NOT's). Perhaps $n = 200$. You claim that the formula is satisfiable; I claim that it is not.
   (a) What do you have to do to convince me that you are right? How much time (in order notation) do you need (as a function of n)? Note: Only count the time needed to actually show me that you are right, not any time needed to figure out how to do so.
   (b) What do I have to do to convince you that I am right? How much time (in order notation) do I need (as a function of n)? Note: Only count the time needed to actually show you that I am right, not any time needed to figure out how to do so.

   Your manager calls you into the office with the following comment:

   > We are now moving into the business of Boolean formula satisfiability. Starting next month, every morning we will be receiving a large number of large Boolean formulas. For each formula, we will need to determine if it is satisfiable. Note that we do not have to actually find the satisfying assignment; we just need a YES/NO answer for each formula.
   >
   > Once again we need your unique skills. You have two weeks to write a lightning fast program to solve satisfiability for these formulas.

9. Of course, you have no idea how to write such a program. You scour the internet but cannot find a satisfactory program to solve satisfiability for Boolean formulas. However, you do find a great program to solve satisfiability for Boolean circuits.

(a) Explain very briefly in English how you would use this Boolean circuit program to solve satisfiability for Boolean formulas.

(b) Assume that the Boolean circuit satisfiability program works in linear time $\Theta(m)$, where m is the number of gates and/or wires in the Boolean circuit. How fast can you determine if a formula with n connectives is satisfiable? Justify.

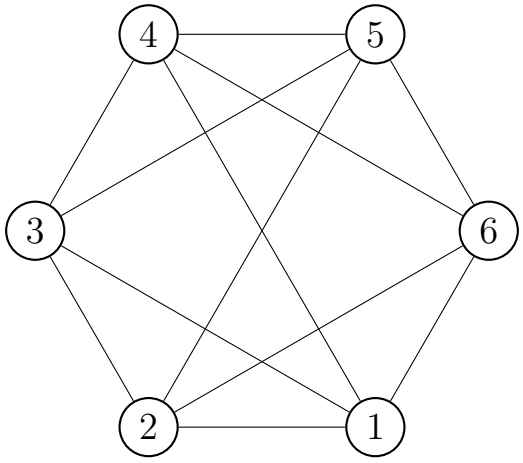
(c) Assume that the Boolean circuit satisfiability program works in time $\Theta(m^r)$, where m is the number of gates and/or wires in the Boolean circuit. How fast can you determine if a formula with n connectives is satisfiable? Justify.

4 Graph Coloring

An undirected graph $G = (V, E)$ is *c-colorable* if each vertex can be assigned a color such that at most c colors are used and no two vertices that share an edge have the same color.

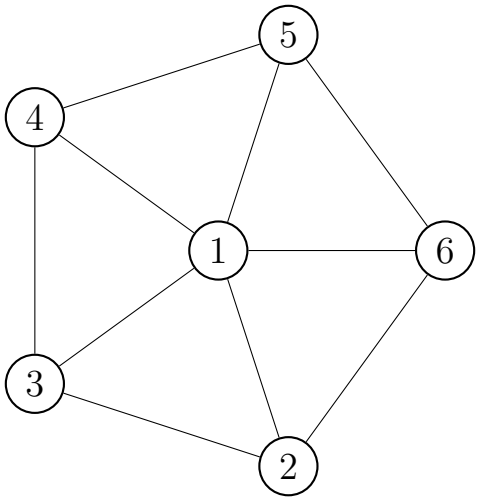
10. You must solve these problems as described. There may be much more clever methods, but we do not care.

    (a) Show that the following graph is 3-colorable, by assigning a color to each vertex and showing that no edge has the same color on its two endpoints.



    The three vertices in triangle (1,2,3) must be different colors. By symmetry, we can set those three colors to Red (R), Blue (B), and Green (G), respectively. This way we can guarantee that there is a unique solution to this problem. Use the two tables provided in the back of this assignment.

    (b) Show that the following graph is not 3-colorable, by showing that each possible assignment of colors to the vertices is not a 3-coloring. This is done by finding an edge in which both



    endpoints have the same color.

There are too many possible colorings to make this practical to do by hand. Once again, the three vertices in the triangle (1,2,3) must be different colors, so by symmetry, we can set those three colors once and for all to, say, Red, Blue, and and Green, respectively. Then we only need to color the remaining three vertices. Use the table in the back of this assignment.

11. Imagine that there is a large undirected graph with n vertices drawn on the whiteboard. Perhaps $n = 200$. You claim that the graph is c-colorable; I claim that it is not.
    (a) What do you have to do to convince me that you are right? How much time (in order notation) do you need (as a function of n and c)? Note: Only count the time needed to actually show me that you are right, not any time needed to figure out how to do so.
    (b) What do I have to do to convince you that I am right? How much time (in order notation) do I need (as a function of n and c)? Note: Only count the time needed to actually show you that I am right, not any time needed to figure out how to do so.

5 Finding Satisfying Assignments

Your manager calls you into the office with the following comment:

> It turns out that not only do we need to determine whether a Boolean formula is satisfiable, but, if so, we need to find a satisfying assignment. Furthermore, the inputs are more complicated than we expected: Boolean formulas can not only have variables, but they can also have TRUE's and FALSE's. So an input might, for example, be
>
> $$((A \wedge (\overline{B} \wedge \text{TRUE})) \vee (\overline{A} \wedge C)) \wedge ((B \wedge \overline{C}) \vee \text{FALSE})$$
>
> Once again we need your unique skills. You have two weeks to write a fast program to determine if a formula is satisfiable, and, if so, to find the satisfying assignment.

You find a program

`satisfiable(`H`)`

that allows TRUE's and FALSE's in the input, H, and returns YES or NO depending on whether the Boolean formula is satisfiable. It runs in time $\Theta(n^r)$, for some constant $r \geq 1$, where n is the number of variables.

You also find a program

`substitute(`H, X, V`)`

that substitutes the value V for variable X in formula H, where V is either TRUE or FALSE, and returns the new formula. It runs in linear time.

12. (a) Show how to use (the Boolean formula satisfiability program) `satisfiable` (and `substitute`) to efficiently find a satisfying assignment. Write the pseudo-code.
    (b) How fast is your algorithm? Justify.

Your manager calls you into the office, compliments you on a great job, and continues:

> It turns out that not only do we need to find a satisfying assignment, but the assignment must minimize the number of variables set to TRUE.

You find a progam

`satisfiable_num(`H`,`k`)`

that allows TRUE's and FALSE's in the input, H, and returns YES or NO depending on whether the Boolean formula is satisfiable with at most k variables set to TRUE. It runs in time $\Theta(n^r)$, for some constant $r \geq 1$, where n is the number of variables.

13. (a) Show how to use (the Boolean formula satisfiability program) `satisfiable_num` (and `substitute`) to find a satisfying assignment that minimizes the number of variables set to TRUE. Write the pseudo-code.

        HINT: eurtottesebtsumselbairavynamwohenimretedtsrif.

        HINT for hint: sdrawkcabtidaer.

    (b) How fast is your algorithm? Justify.

6 Polyomino Tiling

It turns out that polyomino tiling of rectangles is very important to Alpha Beta Gamma's business. A tiling problem is a list of (not necessarily distinct) polyominos $p_1, p_2, \ldots, p_n$, and a board size, $a \times b$, where $a, b \leq n$. The goal is to cover as much of the rectangle as possible with the given tiles. There is no requirement that a tile is small enough to actually fit into the rectangle.

Alpha Beta Gamma does not want to bother solving a tiling problem if the best tiling does not cover enough squares. There are two problems that need to be solved.

(1) Given a list of polyominos $p_1, p_2, \ldots, p_n$, a board size, $a \times b$, where $a, b \leq n$, and a target t, determine if there is a tiling of the rectangle, that covers at least t squares.

(2) Given a list of polyominos $p_1, p_2, \ldots, p_n$, and a board size, $a \times b$, where $a, b \leq n$, find a tiling of the rectangle that covers as many squares as possible.

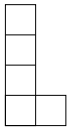
These two problems seem to be hard to solve efficiently. Not surprisingly, your manager asks you to write programs to solve the two problems. As usual you have no idea how to write such programs. For both problems, you find an efficient program on the Internet that solves that problem. Unfortunately your budget will only allow you to buy one such program.

14. Assume that you have a program that solves the *second* problem in time $\Theta(n^j)$, for $j \geq 1$. Can you use it to solve the *first* problem in polynomial time? If so, how, and how fast is your algorithm?

15. Assume that you have a program that solves the *first* problem in time $\Theta(n^k)$, for $k \geq 1$. Can you use it to solve the *second* problem in polynomial time? If so, how, and how fast is your algorithm?

When designing your algorithms, do not worry about how to represent and manipulate polyominos. Without these details (or even with them), you will need to make reasonable, but necessarily slightly vague, arguments to get upper bounds on your run time.[2]

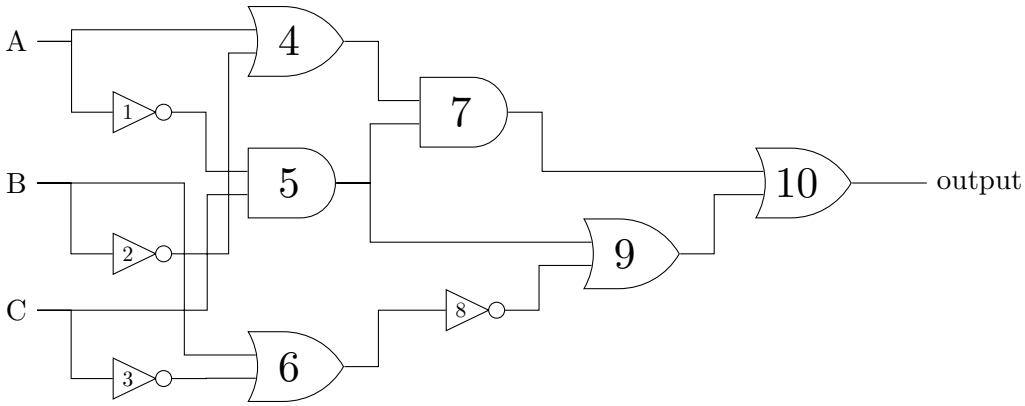
[2]One way to represent a polyomino is as a list of edges (direction, length). For example, the order 4 L-polyomino



could be represented as the list

(up, 4), (right, 1), (down, 3), (right, 1), (down, 1), (left, 2).

Assignment to the inputs $A, B \equiv$ TRUE, $C \equiv$ FALSE.



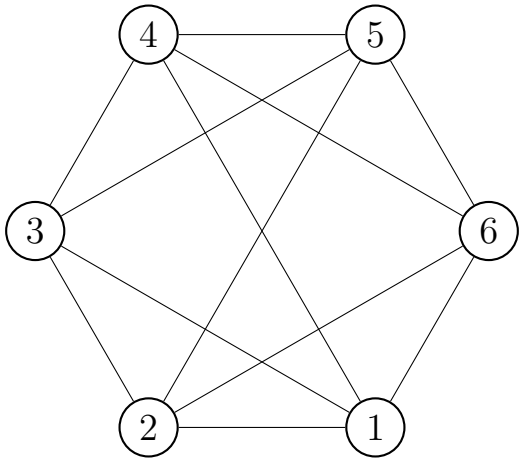
Gate	Output value
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Satisfying assignment: $A, D \equiv$ FALSE and $B, C \equiv$ TRUE.

Clause
$(A \vee B \vee C)$
$(A \vee B \vee \overline{C})$
$(A \vee \overline{B} \vee C)$
$(\overline{A} \vee B \vee \overline{C})$
$(A \vee B \vee \overline{D})$
$(\overline{A} \vee B \vee \overline{D})$
$(A \vee \overline{B} \vee \overline{D})$
$(\overline{A} \vee B \vee D)$
$(A \vee C \vee D)$
$(A \vee C \vee \overline{D})$
$(\overline{A} \vee C \vee D)$
$(\overline{A} \vee \overline{C} \vee D)$
$(B \vee C \vee D)$
$(B \vee \overline{C} \vee D)$
$(\overline{B} \vee C \vee D)$
$(\overline{B} \vee \overline{C} \vee \overline{D})$

Vertex	1	2	3	4	5	6
Color (R,B,G)	R	B	G			

Edge	Color left endpoint	Color right endpoint
(1,2)		
(1,3)		
(1,4)		
(1,6)		
(2,3)		
(2,5)		
(2,6)		
(3,4)		
(3,5)		
(4,5)		
(4,6)		
(5,6)		



123456	Bad edge	Color on endpoints
RBGRRR		
RBGRRB		
RBGRRG		
RBGRBR		
RBGRBB		
RBGRBG		
RBGRGR		
RBGRGB		
RBGRGG		
RBGBRR		
RBGBRB		
RBGBRG		
RBGBBR		
RBGBBB		
RBGBBG		
RBGBGR		
RBGBGB		
RBGBGG		
RBGGRR		
RBGGRB		
RBGGRG		
RBGGBR		
RBGGBB		
RBGGBG		
RBGGGR		
RBGGGB		
RBGGGG		