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

1 Sorting

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

> Alpha Beta Gamma is becoming a big-data company. Starting next month, every morning we will be receiving a large number of large lists of large values. We will need to sort the lists quickly. Each value in a list is an unsigned integer stored in four words of memory; so each value uses 256 bits ($4 \cdot 64$).
>
> 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. You have two weeks to write a lightning fast program to sort these lists.

1. As you leave, you realize that you have no idea how to write such a program because ... [Pick one. If you would like, you can answer for an aquaintance or for a fictional character]:
   (a) You never actually graduated from college. You just pretended to have a degree on your job application.
   (b) You did graduate from college, but you did not major in computer science.
   (c) You majored in computer science, but your algorithms course barely discussed sorting.
   (d) Your algorithms course discussed sorting, but you did not go to class or do the homework.
   (e) You did do the work, but you still did not understand the material.
   (f) You understood the course material, but have long since forgotten it.
   (g) You still remember the course material, but the theoretical material you learned in class has little to do with practical, real life sorting algorithms.
   (h) OTHER: FILL IN THE BLANK.

In any case you have to deal with the situation. You scour the Internet and find a well-regarded sorting package. It is just what you need: It allows you to input a list to be sorted with a parameter for how many words each value uses. You are all set, and can relax for the rest of the month, while pretending you are writing the sorting program.

In order to get exact results, for the remaining problems in this section, ignore the cost of preparing the lists for input into the sorting package or processing the results (assuming those costs are within reason).

2. Assume you are sorting k lists (of four-word values) with sizes (number of values) $n_1, n_2, \ldots, n_k$ where $n = \sum_{i=1}^{k} n_i$.
   (a) Assume that the sorting program takes αwm time to sort a list with m values, where each value consists of w words, for some constant α. How long does it take to sort all k lists. Simplify.

(b) Assume that the sorting program takes αwm^2 time. How long does it take to sort all k lists, where each list has size n/k? Simplify.

(c) In general, assume that the sorting program takes αwm^r time, for some constant r. How long does it take to sort all k lists, where each list has size n/k? Simplify.

3. It turns out that your software license only allows you to run the sorting program once per day.

(a) It is too late to write your own program or find a new software package. What should you do, using the software that you have and being limited by one (large) sort per day? If you do any pre or post processing on the data it must be at most linear time in the worst-case. For example, you cannot use a hash table, because that only has good average-case performance.

(b) If it takes αwm time to sort a list of size m, how long does it take to sort all k lists? Simplify. How does this compare with your previous time from Problem 2a?

(c) If takes αwm^2 time to sort a list of size m, how long does it take to sort the k lists, each of size n/k? Simplify. How does this compare with your previous time from Problem 2b?

(d) In general, if takes αwm^r time to sort a list of size m, how long does it take to sort k lists, each of size n/k? Simplify. How does this compare with your previous time from Problem 2c?

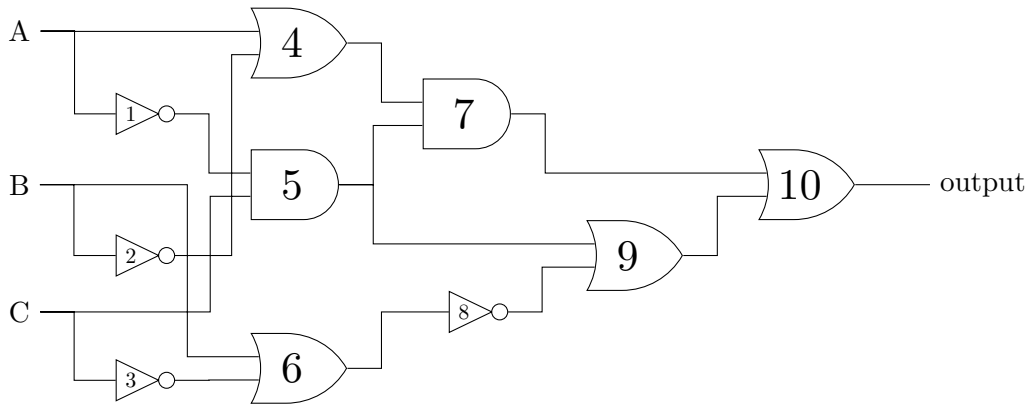
2 Boolean Formula Evaluation

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.

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

> We are now 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.
>
> After you did such an impressive job on the sorting program, we at Alpha Beta Gamma believe that we can rely on you to write a lightning fast program to evaluate these Boolean formulas.

Given the last minute difficulties you had with the sorting program, and your not completely satisfactory solution, perhaps your manager's comment about your "impressive job" was sarcastic. Not being sure, you accept the compliment graciously as if it were real.

Of course, 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? Can you do it in polynomial time? If so describe how. 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? Can I do it in polynomial time? If so describe how. 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.

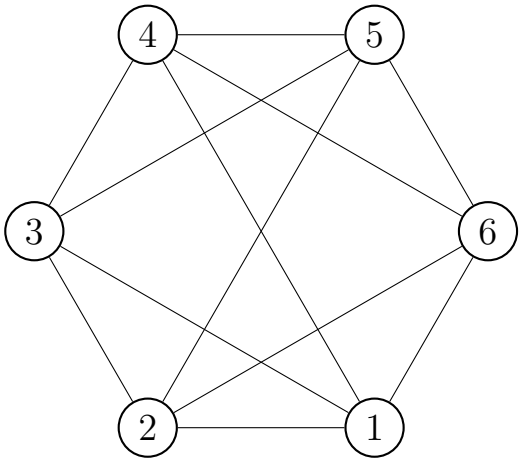
   (c) Assume that the Boolean circuit satisfiability program works in time $\Theta(m^r)$, where m is the number of gates and/or wires in the Boolean circuit. How fast can you determine if a formula with n connectives is satisfiable? Justify.

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.

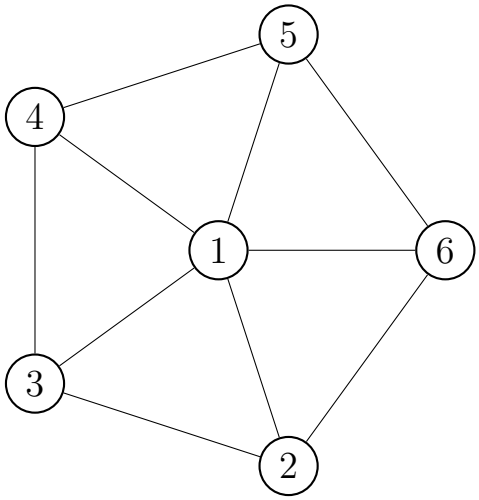
10. You must solve these problems as described. There may be much more clever methods, but we do not care.

    (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 maximize 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 least 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 maximizes 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 Inspecting Hallways

Alpha Beta Gamma needs to hire a security guard to make rounds through the hallways of its office building. A round is a circuit that starts at the security center, passes over each hallway at least once, and returns to the security center. A round has an upper bound on how long it can take.

For each applicant, the hiring committee first determines if there is some circuit that is short enough, given how fast the applicant walks. Any applicant that does not walk fast enough is unacceptable, and is immediatately rejected. After all of the candidates have been interviewed, the best acceptable applicant is hired based on walking speed and other important qualities. Then an optimal (shortest) circuit is found for the new employee.

There are two problems that need to be solved. They can both be represented by a weighted, undirected graph, with positive weights on the edges, where the vertices represent the hallway intersections and the edges represent the hallways.

(1) Given a weighted, undirected graph $G = (V, E, W)$, and a target length L, determine if there is some circuit that crosses every edge at least once with total length at most L.

(2) Given a weighted, undirected graph $G = (V, E, W)$, find a circuit that crosses every edge at least once with minimum total length.

These two problems seem to be hard to solve efficiently. Not surprisingly, your manager asks you to write programs to solve them, and, as usual, you have no idea how to do so. 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. Prove that there is a circuit that crosses every edge at most twice.

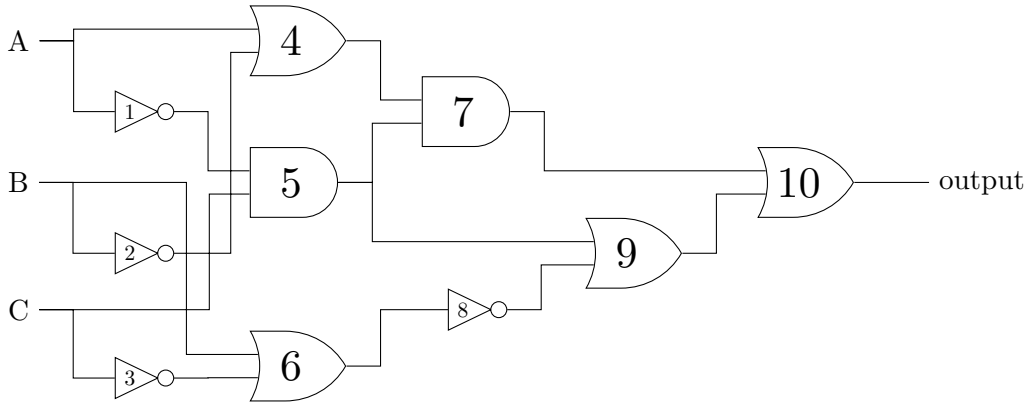
16. Assume that you have a program that solves the *first* problem in time $\Theta(n^k)$, for $k \geq 1$. Can you use it to solve the *second* problem in polynomial time? If so, how, and how fast is your algorithm?

17. Show that Problem (1) is in the class **NP.**

NOTES:

- The actual speed of the applicants can be ignored, since that can be taken into account by just considering the length of the circuit.
- You can assume that the security center is at an intersection (i.e. vertex), since, even if the security center is on a hallway, you can create a new vertex in the graph to represent the location of the security center.
- For simplicity, you can assume that the weights on the edges are integers from 1 to n, where n is the number of vertices in the graph.

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



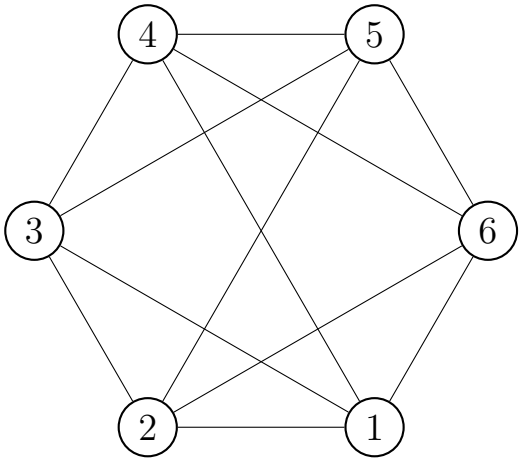
Gate	Output value
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

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		