**Разбор лабы по булевым функциям**

**А. Форма Крома.**

КНФ в форме Крома - — конъюнкция выражений в скобках, каждое из которых представляет собой дизъюнкцию нескольких литералов, число которых не превышает двух. Легко понять, что каждую такую скобку можно расписать через импликацию как (\overline a \to b \wedge \overline b \to a). Таким образом, формула сводится к конъюнкции импликаций и мы имеем право построить граф импликаций, то есть для каждой вершины I = 1 … n добавить в граф ребро (-a, b) и (-b, a). Задача сводится к проверке: правда ли всегда в одной из скобочек мы получим 0? Ответ на вопрос мы получим, если поймем: правда ли, что хотя бы для одной вершины существует путь из !а в а и из а в !а одновременно. Эту задачу можно решить разными способами:

- запуститься от каждой вершины (и их отрицаний) ДФСом и добавить все транзитивные ребра. Затем проверить, что хотя бы одной вершины есть путь из !a в а и из а в !а.

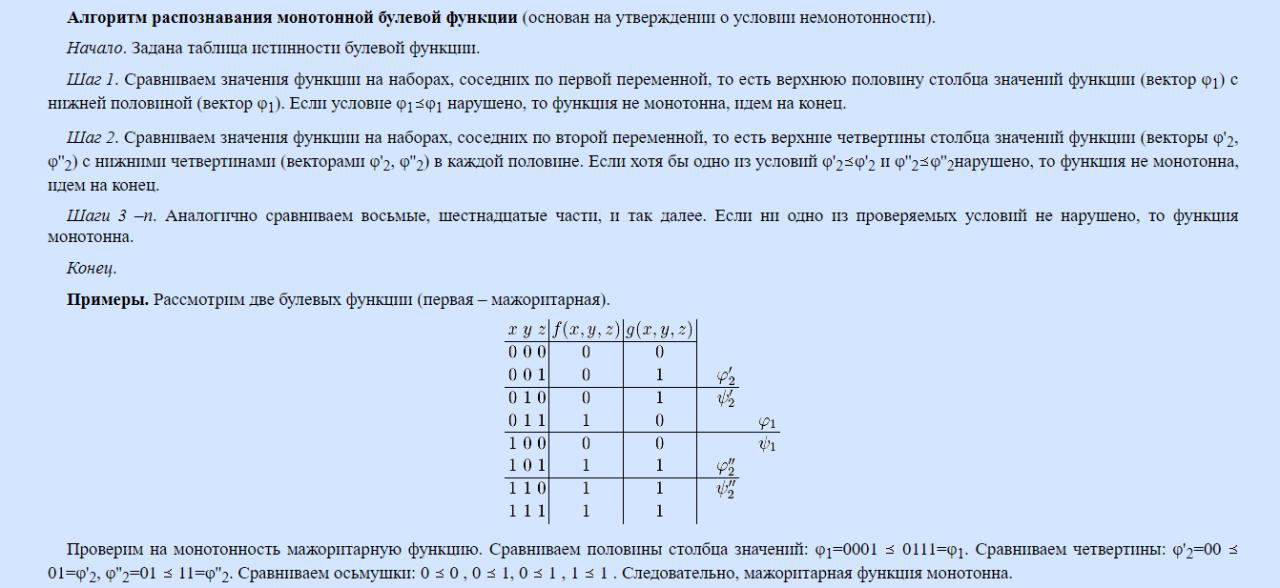
- Пусть вершина = А. Запустимся от каждой А ДФСом и проверим, используя used, что дошли до !А. Запустимся от !А и проверим, что дошли до А (так как граф ориентированный). Если оба раза проверки дали true, значит, функция – тождественный ноль.  
  
Возможные ошибки :

- забыть обнулять used после каждого запуска дфс.  
- не заметить, что возможны отрицательные индексы так как !а = -а. Надо сделать сдвиг по индексам минимум на 15.

**B. Функция Хорна**

КНФ в форме Хорна— это конъюнкция выражений в скобках, каждое из которых представляет собой дизъюнкцию литералов, в которой присутствует не более одного литерала без отрицания. Очевидно, что функция будет тождественным нулем, если хотя бы одна из скобочек обратится в ноль. А она обратится в ноль, если все литералы внутри скобки окажутся нулями. Воспользуемся алгоритмом, описанном в вики-конспектах.   
 Пусть изначально все переменные у нас не определены – это так, потому что в каждую из них мы можем поставить как 0, так и 1. Найдем одиночно стоящие переменные. Очевидно, что в них нужно поставить 1, так как это худший для нас случай. Присвоим всем таким переменным значение 1, если переменная входит без отрицания и 0 иначе, так как в конъюнкции они должны дать 1. Таким образом, мы определили значение одиночной переменной. А значит, мы можем однозначно определить значение итой переменной в других скобках (I – индекс найденной одиночной переменной). Соответственно, после этого в скобках, в которых мы определили значение итой переменной, могли появиться новые одиночные переменные. Если это так, повторим алгоритм. Когда одиночные переменные перестали появляться, нам осталось лишь сделать проверку: если хотя бы в одной скобочке значение получилось нулю, ответ YES, иначе – NO. (Заметим, что если после работы алгоритма значение скобки однозначно определить нельзя, она автоматически приравнивается единице).

**С. Полный набор**

Заметим, что набор является полным, если в нем есть хотя бы одна функция, не сохраняющая 0, хотя бы одна, не сохраняющая 1, хотя бы 1 немонотонная, хотя бы 1 нелинейная, хотя бы 1 несамодвойственная. Остается лишь это проверить. Сохраняющая ноль – если строка значений начинается с 0. Сохраняющая 1 – если строка значений заканчивается на 1. Самодвойственна функция только в том случае, если строка ее значений симметрична относительно середины. Функция линейна, если в полиноме Жегалкина строка, которой соответствует значение полинома 1, содержит не более 1 единицы. Функция монотонна если большему входному набору соответствует неменьшее значение функции. Проверить можно следующим способом : 

Опасные места: Если количество переменных функции равна 0

**D. К или Д**

Это тот случай, когда решение кроется в названии. Оптимальное решение основывается на представлении числа s в качестве скнф или сднф, где литералы – числа от 1 до n. Это можно сделать, если устроить побитовую проверку (очевидно, как – если итый бит в s равен 1, то все числа от 1 до n, в которых итый бит равен 1 брать без отрицания, в которых 0 – с отрицаниемы). Есть и второй, более сложный в реализации, но более простой по идеи алгоритм – полный перебор, в котором нужно перебрать 3 вещи: нужно ли брать отрицание переменной или нет, какие функции соединяют переменные, как расставить скобочки. Но при этом с большой вероятностью нужно делать отсечение по времени, кроме того – реализовать сложно.

Отвечу на вопрос: почему в четвертом тесте ответ impossible. Ответ не может быть равен 1|!1, так как предполагается, что у нас не ровно 32 бита, а сколько угодно.

Опасное место: нужно аккуратно следить за тем, чтобы не было переполнения, поэтому лучше всего использовать 64-ый бит

**E.**

Алгоритм вычисления глубины схемы очевиден. Если узел – лист, его глубина ноль, в противном случае его глубина – максимум из глубины его листов + 1.

Алгоритм решения также тривиален – нам известны значения на листах. Все, что остается сделать – построить зависимость от выхода до листов – это можно сделать на вводе. То есть для каждого узла мы должны запомнить узлы, от значения которых зависит значение данного.

Остается построить рекурентную формулу =)) теперь мы знаем, от каких узлов зависит решение функции и знаем саму зависимость – остается только реализовать рекурсивную функцию )

**F.**

Решение очевидно. Решать методом треугольника. Отлично описано на хабре.