Задание 7

Описание формата входа и выхода

① Вход

Моя программа принимает на вход хорновскую формулу, которой предшествует кол-во переменных в ней и сами переменные, которые представляются строкой вида: первая переменная; 1 пробел; вторая переменная; 1 пробел; ...; последняя переменная. Хорновская формула представляется импликациями и отрицательными дизъюнктами, состоящими из дизъюнкций произвольного количества отрицаний. Импликациям предшествует их количество. Сама импликация представляется строкой вида: первая переменная; 1 пробел; вторая переменная; 1 пробел; ...; символ '>'; 1 пробел; правая переменная импликации; символ ';', т.е. признак конца строки. Дизъюнкциям предшествует их кол-во. Дизъюнкция представлена строкой вида: первая переменная; 1 пробел; вторая переменная; 1 пробел; ...; символ ';', т.е. признак конца строки. Пример входа (смотрите слева, справа отображена соответствующая хорновская ф-ла):

```
4
x y z w
2
w y z > x ;
> x ;
2
w x y ;
z ;
```

$$\varphi = (w \wedge y \wedge z \to x) \wedge (\to x) \wedge (\bar{w} \vee \bar{x} \vee \bar{y}) \wedge \bar{z}$$

② Выход

Моя программа выводит либо сообщение "UNSAT", в случае невыполнимости хорновской формулы, либо выполняющий её набор значений. Пример выхода программы (для входных данных из примера выше):

```
Horn Sat:

x : false
y : true
z : false
w : false
```

Псевдокод алгоритма

```
ф-я hornSat (Ψ) {
// Вход: хорновская формула Ψ, представленная:
// • контейнером переменных 'vars', где для каждой переменной в 'vars'
// ключом 'key' является её название, а значение 'value' по этому ключу -
// её значение true или false, изначально все false (vars[key] = value);
// • импликациями 'implications', где каждая импликация в 'implications'
// представлена списком переменных 'left' слева от знака '→' и, соответ-
// ствующей этому списку переменной 'right' справа от знака '→';
// • отрицательными дизъюнктами 'disjuncts', состоящими из дизъюнкций произ-
// вольного кол-ва отрицаний, которые представлены списком переменных.
// Выход: выполняющий Ψ набор значений, в случае выполнимости
// Ψ, или сообщение "UNSAT", в обратном случае.
// Примечание: стэк emptyImplLeftElem создается для импликаций, которые
// не содержат переменных в левой части. При наличии таких импликa-
// ций, в emptyImplLeftElem будут помещаться их правые переменные.
    инициализация стэка emptyImplLeftElem;
    для каждой импликации impl в implications:
        если impl.left пуст:
            emptyImplLeftElem.push(impl.right);
    пока emptyImplLeftElem не пуст:
        trueVar := emptyImplLeftElem.top();
        emptyImplLeftElem.pop();
        vars[trueVar]:= true;
        для каждой импликации 'impl' в implications:
            для каждой переменной 'v' в impl.left:
                если v == trueVar:
                    убрать 'v' из impl.left;
                    если impl.left пуст:
                        emptyImplLeftElem.push(impl.right);
                    выход из цикла;
```

```
        для каждой дизъюнкции 'd' в disjuncts:
            count := 0;
            для каждой переменной 'v' в ' ':
                если vars[v] == false:
                    выход из цикла;
                count++;
                если 'count' равен кол-ву 'd':
                    вывод сообщения "UNSAT";
                    return;
        вывод vars;
}
```

Оценка временной сложности

] I – кол-во импликаций, N - суммарное кол-во переменных во всех импликациях. В худшем случае за одну итерацию в стэк может попасть одна "пустая" импликация. Изначально там летит одна "пустая" импликация, как самый худший случай. Тогда временная сложность алгоритма – это $O(N \cdot I^2)$. Т.к. I и N – не более, чем длина формулы => временная сложность от длины входной строки-формулы – $O(L^2)$, где L – длина строки-формулы.

Код программы

```
#include <iostream>
#include <vector>
#include <stack>
#include <list>
#include <map>

class HornFormula
{
public:
  HornFormula(int n, std::map<char, bool>& vars);
  void addImplication(std::pair<std::list<char>, char> &impl);
  void addDisjunction(std::list<char> &disjunction);
  void printSolution();
  void hornSat();
private:
  int n_;
  std::map<char, bool> vars_;
  std::vector<std::pair<std::list<char>, char>> implications_;
  std::vector<std::list<char>> disjunctions_;
};

HornFormula::HornFormula(int n, std::map<char, bool>& vars)
{
  this->n_ = n;
  this->vars_ = vars;
}

//Добавление импликации в хорновскую формулу
void HornFormula::addImplication(std::pair<std::list<char>, char>
&implication)
{
  implications_.push_back(implication);
}

//Добавление дизъюнкции в хорновскую формулу
void HornFormula::addDisjunction(std::list<char> &disjunction)
{
  disjunctions_.push_back(disjunction);
}

//Фукнция проверяющая выполнимость хорновской формулы
void HornFormula::hornSat()
{
  //стэк для элементов справа от  '->' импликаций, в которых нет элементов
  //слева от '->'
  std::stack<char> emptyFirstElemImpl;

  for (auto i: implications_)
  {
    if (i.first.empty())
    {
      emptyFirstElemImpl.push(i.second);
    }
  }
  while (!emptyFirstElemImpl.empty()) //пока есть не выполненная импликация
  {
    char trueVar = emptyFirstElemImpl.top();
    emptyFirstElemImpl.pop();
    vars_[trueVar] = true; //присваивание переменной справа значения true
```

```
    for (auto& i: implications_)
    {
      for (auto j = i.first.begin(); j != i.first.end(); j++)
      {
        if (*j == trueVar)
        {
          i.first.erase(j);
          if (i.first.empty())
          {
            emptyFirstElemImpl.push(i.second);
          }
          break;
        }
      }
    }
  }

  for (auto i: disjunctions_)
  {
    int count = 0;

    for (auto j: i)
    {
      if (vars_[j] == false)
      {
        break;
      }

      count++; //количество положительных переменных дизъюнкции

      if (count == i.size())
      {
        std::cout << "\nUNSAT\n";
        return; //формула невыполнима
      }
    }
  }
  printSolution();
  return;
}

//Функция вывода решения в консоль
void HornFormula::printSolution()
{
  std::cout << "\nHornSat:\n";
  for (auto i: vars_)
  {
    std::cout << i.first << ": ";
    if (i.second == true)
    {
      std::cout << "true" << '\n';
    }
    else
    {
      std::cout << "false" << '\n';
    }
  }
}
```

```
int main()
{
  std::map<char, bool> elements;

  int nElements = 0;
  char elem;
  std::cin >> nElements;

  for (int i = 0; i < nElements; i++)
  {
    std::cin >> elem;
    elements[elem] = false;
  }

  HornFormula hornFormula(nElements, elements);

  int nImlications = 0;
  std::cin >> nImlications;

  for (int i = 0; i < nImlications; i++)
  {
    std::cin >> elem;
    std::pair<std::list<char>, char> implication;

    while (elem != '>')
    {
      implication.first.push_back(elem);
      std::cin >> elem;
    }

    std::cin >> elem;

    while (elem != ';')
    {
      implication.second = elem;
      std::cin >> elem;
    }

    hornFormula.addImplication(implication);
  }

  int nDisjunctions = 0;
  std::cin >> nDisjunctions;

  for (int i = 0; i < nDisjunctions; i++)
  {
    std::cin >> elem;
    std::list<char> disjunction;

    while (elem != ';')
    {
      disjunction.push_back(elem);
      std::cin >> elem;
    }
    hornFormula.addDisjunction(disjunction);
  }

  hornFormula.hornSat();

  return 0;
}
```

Тесты

Входные данные	Выходные данные (с пояснением)
$\varphi = (w \wedge y \wedge z \rightarrow x) \wedge (x \wedge z \rightarrow w) \wedge (x \rightarrow y) \wedge (\rightarrow x) \wedge (x \wedge y \rightarrow w) \wedge (\bar{w} \vee \bar{x} \vee \bar{y}) \wedge \bar{z}$	Начинаем со всех значений false и далее видим, что ‘x’ должно быть true в силу импликации ‘$\rightarrow$x’. Импликация ‘x$\rightarrow$y’ вынуждает присвоить ‘y’ значение true. Т.к. ‘x’ и ‘y’ – true, импликация ‘x$\wedge$y$\rightarrow$w’ вынуждает присвоить ‘w’ значение true. Т.о. дизъюнкт ‘$\bar{w} \vee \bar{x} \vee \bar{y}$’ ложен $\Rightarrow \varphi$ невыполнима.
4 x y z w 5 w y z > x; x z > w; x > y; >x; x y > w; 2 w x y; z;	UNSAT
$\varphi = (w \wedge y \wedge z \rightarrow x) \wedge (x \wedge z \rightarrow w) \wedge (x \rightarrow y) \wedge (\rightarrow x) \wedge (x \wedge y \rightarrow w) \wedge (\bar{w} \vee \bar{x} \vee \bar{y} \wedge \bar{z})$	Начинаем со всех значений false и далее видим, что ‘x’ должно быть true в силу импликации ‘$\rightarrow$x’. Импликация ‘x$\rightarrow$y’ вынуждает присвоить ‘y’ значение true. Т.к. ‘x’ и ‘y’ – true, импликация ‘x$\wedge$y$\rightarrow$w’ вынуждает присвоить ‘w’ значение true. Т.о. импликация ‘x$\wedge$z$\rightarrow$w’ – истинна, как и дизъюнкт ‘$\bar{w} \vee \bar{x} \vee \bar{y} \wedge \bar{z}$’ $\Rightarrow \varphi$ выполнима.
4 x y z w 5 w y z > x; x z > w; x > y; >x; x y > w; 1 w x y z;	HornSat: w: true x: true y: true z: false
$\varphi = (x \rightarrow y) \wedge (\rightarrow y) \wedge (\bar{x} \vee \bar{y})$	Начинаем со всех значений false и далее видим, что ‘y’ должно быть true в силу импликации ‘$\rightarrow$y’. С выбранным значением для ‘y’ импликация ‘x$\rightarrow$y’ – истинна, как и дизъюнкт ‘$\bar{x} \vee \bar{y}$’ $\Rightarrow \varphi$ выполнима.

2 x y 2 x > y; > y; 1 x y;	HornSat: x: false y: true
$\varphi = (x \to y) \wedge (y \to x) \wedge \bar{x}$	Начинаем со всех значений false и далее видим, что с выбранными значениями все условия формулы истины $\Rightarrow \varphi$ выполнима.
2 x y 2 x > y; y > x; 1 x;	HornSat: x: false y: false

Задание 8

Псевдокод

процедура maxWaitingTime Order (n, t) {

// Вход: количество клиентов n; набор времен обслуживания клиентов

// $t_i : i \in (1, \dots, n)$.

// Выход: последовательность номеров клиентов, которая максимизирует

// суммарное время ожидания клиентов.

сортировать t по убыванию t_i;

вернуть t;

}

Обоснование корректности

] t_1 - максимальное время обслуживания, тогда для некоторых других эл-ов t_i, $t_i < t_1$. Предположим, что выбор t_i вместо t_1 может уменьшить время ожидания T. Тогда, $T_{old} = (n-1)t_1 + (n-i)t_i$, а $T_{new} = (n-1)t_i + (n-i)t_1$. Проверяем, что $T_{old} > T_{new}$:

$$(n-1)t_1 + (n-i)t_i \overset{?}{\geq} (n-1)t_i + (n-i)t_1$$

$$t_1 n - t_1 + t_i n - t_i i \overset{?}{\geq} t_i n - t_i + t_1 n - t_1 i$$

$$-t_1 + t_1 i \overset{?}{\geq} -t_i + t_i i$$

$$t_1(i-1) \overset{?}{\geq} t_i(i-1)$$

$t_1 > t_i$ – верно из усл.

Таким образом, t_1 должно быть максимальным временем обслуживания.

Оценка временной сложности

] $T(n)$ - время работы ф-и maxWaitingTime Order.

Временная сложность сортировки n элементов есть $O(n \log n)$;

Следовательно, получим $T(n) = O(n \log n)$ – время работы ф-и maxWaitingTime Order.