Лабораторная работа «Булевы функции»

1. Реализуйте, на С++, класс булевых функций, представимых таблицей истинности в виде вектора, СКНФ, СДНФ, сокращённой ДНФ, полиномом Жегалкина, картой Карно | диаграммой решений. Класс/объект содержит:
   -**данные:** вектор таблицы истинности, матрицу сокращённой ДНФ, вектор коэффициентов полинома Жегалкина, карту Карно | диаграмму решений;
   -**методы конверсии** таблицы истинности (СКНФ, СДНФ), в сокращённую ДНФ (Квайн, Нельсон), полином Жегалкина, карту Карно… **и обратно**;
   -**методы потокового ввода / вывода** из / в `cin` | `cout` | `*.txt`;
   -**методы вычисления значения** в точке, записи в таблицу истинности;
   -**методы проверки свойств**: самодвойственности, сохранения нуля, единицы, монотонности, линейности, симметричности.
   Используйте побитовые булевы операции `&`, `|`, `^` и сдвиги `<<`, `>>`.
2. Для булевой функции $f(x_1, .. x_n)$ ($8 \leq n \leq 10$), заданной таблично столбцом её значений истинности, - двоичным кодом $(N + 192)^{2\wedge(n-3)}$, где N - номер Вашего варианта, вычислите СКНФ, СДНФ, редуцированную ДНФ, полином Жегалкина, и проверьте свойства f. Ввод / вывод f – из / в *.txt.

Ф.А.: Требования к отчёту:

В отчете 5 обязательных пунктов (как и в первой лаб. р.):

1. Постановка задачи: какие ограничения накладываются и почему.

2. Решение задачи: код программы, можно - реализацию конверторов и методов проверки свойств, **только** (С.В.).

3. Обоснование решения: доказательство правильности + (для булевых функций!) **анализ ёмкостной сложности**.

4. Вычислительный эксперимент: что с этим пакетом было сделано, и что получилось.

5. Выводы: не только списанные из учебника, но и выстраданные при программировании.

Дополнительные пояснения

1. Защищённые / приватные данные класса булевых функций (БФ) содержат **ВСЕ** формы представления функции, из которых, при инициализации объекта, параметром конструктора задаётся одна, а остальные, при инициализации, получаются из неё - конверсией. Инициализация каждого из представлений – передачей значения параметра (представления БФ). При этом полиморфные конструкторы должны либо иметь прототипы, взаимно однозначно соответствующие виду представления, в котором передаётся инициализирующее значение представления БФ либо принимать параметр, задающий тип представления БФ. Пример:
```
typedef enum BFinitType {VTab, PDNF, PCNF, RDNF, ZheP, Karn};
BF(byte n, const uint32*, BFinitType = VTab);
```
2. Поскольку носитель БФ однозначно кодирует её СДНФ, а его дополнение – СКНФ, 2-мя разными способами интерпретации, можно [не] дублировать носитель/дополнение – на усмотрение программиста.
Кроме **вектора истинности**, **необходимы:**
- **массив 2^n двоичных коэффициентов полинома Жегалкина** (ПЖ), индексированный n-мерными двоичными векторами мульти-степеней мономов, **или** список/массив двоичных векторов мульти-степеней мономов, входящих в ПЖ с 1-коэффициентами;
- **матрица RDNF** (сокращённой ДНФ): каждая конъюнкция RDNF описывается парой 2-ичных векторов: вектор \`a\` вхождения, и вектор \`b\` инвертирования булевых переменных конъюнкции;
$RDNF[f](x) = \vee_{j=0..m-1} \&_{i=0..n-1} (d_{i,j} \Rightarrow (x_i \sim b_{i,j}))$,
где i – индекс переменной в векторе,
j – индекс в RDNF конъюнкции подмножества переменных x_i, отвечающих $d_{i,j} = 1(true)$ (вхождение x_i в j-ю конъюнкцию RDNF), инвертированных при $b_{i,j} = 0$ (инверсия x_i в j-й конъюнкции RDNF),
m – число конъюнкций в RDNF.
- **матрица Карно**.

3. Методы `>>`, `<<` потокового ввода/вывода – для чтения/записи из/в поток(а) объекта БФ, заданной **таблицей истинности**, с инициализацией (при чтении) остальных её представлений. Поток может быть подключён к txt-файлу. Для ввода/вывода объекта БФ из/в поток(а) (консоль/txt-файл), в формате любого из **остальных представлений БФ**, - **специальные** методы, например, ZhFromStream | ZhToStream, RDNFfromStream | RDNFtoStream, с 2-мя параметрами: указатель на поток, [ссылка на] объект БФ. При чтении БФ в одном из представлений, остальные её представления инициализируются конверторами (из) таблицы истинности.
4. Методы **записи** значений **в таблицу истинности** – **приватные**, используются только в конверторах и конструкторах БФ.
   Методы **вычисления значений БФ**, заданной одним из её представлений, - в точке $x \in \mathbf{bool}^n$, - **публичные**. Для каждого представления БФ, отвечающего члену данных, – свой метод вычисления значения БФ в точке: ValT, ValRDNF, ValZh, ValKarn. Контроль правильности класса – совпадение значений в случайных точках - для всех представлений БФ.
5. Для каждого такого представления БФ – свой формат данных: см. п.2.
6. Булевы векторы точек $x \in \mathbf{bool}^n$, вхождения/инвертирования переменных x_i представляются unsigned int. Это позволяет, **в пределах RAM PC(!!!)**, работать с $n = \mathbf{0 .. nMax} \leq 32$ переменных. **Практически** Вы опробуете:
   $n = 8$ – для печати карт Карно (16 × 16), и
   $n = 8 .. 10$ – для оценок временной/емкостной сложности методов.
7. Формула индекса $j(x)$ точки $x \in \mathbf{bool}^n$ в таблице истинности БФ – проще в LSBF (big-endian) кодировке точки x: $j(x) = sum(x_i \cdot 2^i, i = 0 .. n\text{-}1)$. Аналогично, вектор-столбец таблицы истинности БФ удобнее перечислять «сверху-вниз»: $x = 0 .. (2^n – 1)$, считая верхние биты младшими (Least Significant Bits). Такая интерпретация вектора истинности дана в примере упражнения. Однако, операции <<, >>, ++ сдвигов, инкремента, и hex-кодировка байт - интерпретируют uint-значения в MSBF (little-endian) кодировке uint. Вы можете выбирать «выгодные», с Вашей точки зрения,

последовательность записи бит вектора истинности БФ, и индексацию булевых переменных x_i в uint-представлении точек uint $x \in$ ***bool***^{n}. Пара взаимно-обратных функций показывает «выгоду» этого представления:

```
uint Grey(uint Ind) { return Ind ^ Ind >> 1; } // Ф.А.Новиков «ДМ» Т.1.3.4
uint Grey2Int(uint G) { uint d, m, b; // Most Significant Bit xOR-Accumulator
    for (m = 1U << 31; ~(G & m) & m; m >>= 1); // Most Significant Bit of G
    for (d = b = 0; m; b ^= G & m, d ^= b, b >>= 1, m >>= 1); //!!! C/C++
    return d; // Index of G
} // Grey2Int Ф.А.Новиков «ДМ» А.3.7. Grey∘Grey2Int=Id=Grey2Int∘Grey!
```

8. При $n \leq 32$, можно использовать в конструкторе БФ по таблице истинности:

   typedef unsigned char byte;

   BF(byte n, const bool*); // массив 2^n значений БФ;

   кроме того, можно хранить до 32 смежных последовательных двоичных значений в одном слове uint32, это в 32/4 = 8 раз сократит размер массива:

   BF(byte n, const uint32*); // массив 2^n значений БФ, **упакованных по 32**.

9. Если использовать представление точек $x \in bool^n$ - одним словом uint32, и определить перечислительный тип
   typedef enum BFinitType {VTab, PDNF, PCNF, RDNF, ZheP, Karn};
   то СДНФ, СКНФ, РДНФ, ПЖ, Карно можно инициализировать конструктором с 2|3|4-мя параметрами:
   BF(byte n, const uint32*, BFinitType = VTab, byte m = 0);
   // default: Table of truth Values Packed to uint32[]
   при этом, индексируя точки: $bool^n \ni x \mapsto j(x) = sum(x_i \cdot 2^i, i = 0 .. n\text{-}1)$, и функции: $f \mapsto SuppInd(f) = sum(2^{j(x)}, x \in sup(f) = \{ x \in bool^n \mid f(x) = true\})$,
   $f \mapsto SuppComplemInd(f) = sum(2^{j(x)}, x \in bool^n \backslash sup(f)=\{x \in bool^n \mid f(x)=false\})$:
   byte n = 2;
   uint32 AndValTab[1] = {0x8}, // &
   // $PDNF[f](x) = \vee_{b \in supp(f)} \&_{i=0..n-1} (x_i \sim b_i)$;
   AndPDNF[1] = {0x8}, // = {*SuppInd(&)*} // *AndSupp[1] = {0x3}*,
   // $PCNF[f](x) = \&_{b \in (bool\wedge n)\backslash supp(f)} (\vee_{i=0..n-1} (x_i \nsim b_i))$;
   AndPCNF[1] = {0x7}, // = {*SuppComplemInd(&)*}
   // AndSuppComplem[3] = {0x0, 0x1, 0x2},
   // $RDNF[f](x) = \vee_{j = 0 .. m-1} \&_{i=0 .. n-1} (d[i, j] \Rightarrow (x[i] \sim b[i, j]))$, d – *вектор вхождения*;
   AndRDNF[2] = {0x3, 0x3}, // $RDNF[\&](x_0, x_1) = x_0 \& x_1$, $d = (true, true) = b$;
   // $ZheP[f](x) = \nsim_{d \in bool\wedge n} \&_{i=0 .. n-1} (d_i \Rightarrow x_i)\&c_d, (c_d \in bool)$;
   AndZheP[1] = {0x8}, // AndZhePcoeffSupp[1] = {0x3}, // $ZheP[\&] = x_0 \& x_1$;

OrValTab[1] = {0xE}, // *PDNF[*$\vee$*]*$(x_0, x_1) = x_0 \,\&\, \neg x_1 \vee \neg x_0 \,\&\, x_1 \vee x_0 \,\&\, x_1$;
OrPDNF[1] = {0xE}, // OrSupp[3] = {0x1, 0x2, 0x3},
OrPCNF[1] = {0x1}, // OrSuppComplem[1]={0x0}, //*PCNF[*$\vee$*]*$(x_0, x_1)=x_0 \vee x_1$;
OrRDNF[4] = {0x1, 0x1, 0x2, 0x2}, // *RDNF[*$\vee$*]*$(x_0, x_1) = x_0 \vee x_1$;
// $d(x_0) = 0x1 = b(x_0)$, $d(x_1) = 0x2 = b(x_1)$; //*векторы вхождения/инверсии*
// OrZhePcoeffSupp[3]={0x1, 0x2, 0x3}, // *ZheP[*$\vee$*]*$(x_0, x_1) = x_0 \not\sim x_1 \not\sim x_0 \,\&\, x_1$;
//$(d(x_0)=0x1, d(x_1)=0x2, d(x_1\&x_1)=0x3) \mapsto IndOrZhePcoeffSupp=2^1+2^2+2^3=11$;
OrZheP[1] = {0xB}, // {11}
// $\Rightarrow$:
ImpValTab[1]={0xD}, //ImpSupp[3]={0x0,0x2,0x3}, ImpSuppCompl[3]={1},
ImpPDNF[1] = {0xD}, // *PDNF[*$\Rightarrow$*]*$(x_0, x_1) = \neg x_0 \,\&\, \neg x_1 \vee \neg x_0 \,\&\, x_1 \vee x_0 \,\&\, x_1$;
ImpPCNF[1] = {0x2}; // *PCNF[*$\Rightarrow$*]*$(x_0, x_1) = \neg x_0 \vee x_1$;
ImpRDNF[4] = {0x1, 0x0, 0x2,0x2}; // *RDNF[*$\Rightarrow$*]*$(x_0, x_1) = \neg x_0 \vee x_1$;
// $d(\neg x_0) = 0x1$, $b(\neg x_0)$=0x0, $d(x_1) = 0x2 = b(x_1)$; // *вхождение / инверсия*
// ImpZhePcoeffSupp[3]={0x0, 0x1, 0x3},// *ZheP[*$\Rightarrow$*]*$(x_0, x_1)=1 \not\sim x_0 \not\sim x_0 \,\&\, x_1$;
//$(d(1)=0x0, d(x_0)=0x1, d(x_1\&x_1)=0x3) \mapsto IndOrZhePcoeffSupp=2^0+2^1+2^3=9$;
ImpZheP[1] = {0x9};
// BF Initialization:
BF AND(n, AndValTab), OR(n, OrValTab), IMP(n, ImpValTab);
// ^ initialize `AND`, `OR`, `IMP` by Table of truth Values Packed to uint32[]
BF And(n, AndPDNF, PDNF), Or(n, OrPDNF, PDNF),
Imp(n,ImpPDNF,PDNF); // initialize `And`, `Or`, `Imp` by PDNF
BF and(n, AndPCNF, PCNF), or(n, OrPCNF, PCNF), imp(n,ImpPCNF,PCNF);
// ^ initialize `and`, `or`, `imp` by PCNF
BF andZ(2, AndZheP, ZheP),
//^initialize `andZ` by array of Monomials MultyDegree for Zhegalkin Polynom
orZ(n, OrZheP, ZheP),
//^initialize `orZ` by array of Monomials MultyDegree for Zhegalkin Polynom
impZ(2, ImpZheP, ZheP),
// ^ init. `impZ` by array of Monomials MultyDegree for Zhegalkin Polynom
BF andR(n, AndRDNF, RDNF, 1), orR(n, OrRDNF, RDNF, 2),
impR(n, ImpRDNF, RDNF, 2); // initialize `andR`, `orR`, `impR` by RDNF.

10. **НЕ предполагается** строкового представления формул (с использованием символов переменных и операций) для каких либо форм, и их **парсинга (синтаксического разбора)**! Функции ValSDNF и ValSKNF можно не вводить, ибо они, сводятся к выборке значения ValT[x] по индексу. Функции ValRDNF, ValZhP, ValKarn надо писать, учитывая соответствующее представление функции массивом/матрицей (см. 9. примеры инициализации).