# 1 Построение в виде таблицы истинности

## 1.1 (псевдо)случайной булевой функции от заданного числа переменных;

Просто генерируем вектор длины $2^n$ с помощью генертора случайных чисел

```
GetRandomBF[n_] := Table[Random[Integer, {0, 1}], {2^n}];
```

## 1.2 (псевдо)случайной булевой функции заданного веса от заданного числа переменных

Генерируем вектор из нулей длины $2^n$ и с помощью генератора случайных чисел генерируем индекс и ставим на его место 1 пока не получим $w$ единиц

```
GetRandomBFweight[n_, w_] :=
    Module[{tmp, counter},
      tmp = Table[0, {2^n}];
      counter = 0;
      While[counter < w,
        ind = Random[Integer, {1, 2^n}];
        If[
          tmp[[ind]] == 0,
          tmp[[ind]] = 1;
          counter++;, Continue[]];
        ];
      tmp
      ];
```

## 1.3 (псевдо)случайной линейнойбулевой функции от заданного числа переменных

Как и выше, генерируем случайный вектор длины $n$ соответствующий присутствию(отсутсвию) переменной $x_i$ в полиноме Жегалкина и дальше считаем его таблицу истинности.

```
ClearAll[GetRandomLinearBF];
GetRandomLinearBF[n_] :=
    Module[{ft, tmp, i},
      f = Table[Random[Integer, {0, 1}], {n}];
      tmp = Table[0, {2^n}];
      For[i = 0, i < 2^n, i++,
        If[EvenQ[IntegerDigits[i, 2, n].f],
            tmp[[i + 1]] = 0;,
            tmp[[i + 1]] = 1;
            ];
        ];
```

```
  tmp
  ];
```

Для пункта с линейными приближениями была также создана функция `GetLinearBF[f_, n_]`, которая по вектору $(a_1, ..., a_n)$ возвращает таблицу истинности.

Проверка того, что функция и правда получается линейной будет в пункте 2.2.

# 2 Разработка способов и реализация средствами САВ "Mathematica" преобразований представлений булевых функций

## 2.1 из многочлена Жегалкина в АНФ

Я так и не осознал как мне в принципе задавать функцию через многочлен. Максимум что мне пришло в голову это сделать что-то вроде

```
BooleanFunction[f_, n_] := Function[u, f[[FromDigits[Reverse[u], 2] + 1]]];
```

Где $f$ - Таблица истинности. Однако с этим неудобно работать/либо я просто не придумал как с этим работать.

## 2.2 из таблицы истинности в многочлен Жегалкина и АНФ

Реализуем преобразование из таблицы истинности в АНФ с помощью рекурсивного алгоритма Так как

$$f(\vec{x}) = a0 + a1 * x1 + a2 * x2 + a3 * x1x2 + ... = \sum_{\vec{\alpha} \in \{0,1\}^n} a_f(\vec{\alpha}) * \zeta(\vec{\alpha}, \vec{x})$$

Получаем матрицу преобразования из АНФ в таблицу истинности:

$$Z_n = \begin{pmatrix} Z_{n-1}, & 0 \\ Z_{n-1}, & Z_{n-1} \end{pmatrix}, Z_1 = \begin{pmatrix} 1, 0 \\ 1, 1 \end{pmatrix}$$

Получаем рекурсивный алгоритм для вычисления произведения $Z_n * \vec{a_f}$

Пусть $a_f^{(1)}$ - первые $2^{n-1}$ бит $a_f$, а $a_f^{(2)}$ - вторые. Тогда $Z_n * \vec{a_f} = (a_f^{(1)}, a_f^{(1)} \bigoplus a_f^{(2)})$.

А так как в поле по модулю 2

$Z_n = Z_n^{-1}$, то то же преобразование работает и в обратную сторону.

```
TransFormVec[{0}, 0] = {0};
TransFormVec[{1}, 0] = {1};
TransFormVec[vec_, n_] := TransFormVec[vec, n] =
```

```
      Module[{tmp1, tmp2, x},
        If[Length[vec] != 2^n,
          Return[0, Module]
          ];
        tmp1 = TransFormVec[Take[vec, 2^(n - 1)], n - 1];
        tmp2 = TransFormVec[Take[vec, -2^(n - 1)], n - 1];
        tmp2 = BitXor[tmp1, tmp2];
        Join[tmp1, tmp2]
        ];
```

```
TruthToANF[vec_, n_] := TransFormVec[vec, n];
```

Многочлен Жегалкина же я получаю из АНФ просто составляя строки из бинароного представления $i$ для коэффициента АНФ $a_f[i]$;

## 2.3 из многочлена Жегалкина в таблицу истинности;

Так как многочлен Жегалкина в моём коде - строка, я не смог придумать как его преобразовывать в АНФ нормальным способом.

Поэтому представлена только функция АНФ -> таблица истинности

```
AnfToTruth[vec_, n_] := TransFormVec[vec, n];
```

## 2.4 из таблицы истинности в действительный многочлен

Аналогично пункту 2.2, только теперь мы работаем не в поле по модулю 2 а в действительных числах

$$B_n = \begin{pmatrix} Z_{n-1}, & 0 \\ -Z_{n-1}, Z_{n-1} \end{pmatrix}, B_1 = \begin{pmatrix} 1, 0 \\ -1, 1 \end{pmatrix}$$

Получаем рекурсивный алгоритм для вычисления произведения $B_n * \vec{f}$

Пусть $f^{(1)}$ - первые $2^{n-1}$ бит $f$, а $f^{(2)}$ - вторые. Тогда $B_n * \vec{f} = (f^{(1)}, f^{(2)} - f^{(1)})$.

В итоге получаем коэффиценты действительного многочлена.

```
TransFormPoly[vec_, 1] := {{1, 0}, {-1, 1}}.vec;
TransFormPoly[vec_, n_] := TransFormPoly[vec, n] =
      Module[{tmp1, tmp2, x},
        If[Length[vec] != 2^n,
          Return[0, Module]
          ];
        tmp1 = TransFormPoly[Take[vec, 2^(n - 1)], n - 1];
        tmp2 = TransFormPoly[Take[vec, -2^(n - 1)], n - 1];
        tmp2 = tmp2 - tmp1;
        Join[tmp1, tmp2]
        ];
```

### 2.5 вычисление списка спектральных коэффициентов (Фурье, Адамара-Уолша) по таблице истинности

Для Адамара-Уолша мы берем вектор значений $(-1)^{\vec{f}}$ или то же самое $1 - 2 * \vec{f}$

Умножаем его на матрицу, которая задана рекурсивно:

$$H_n = \begin{pmatrix} H_{n-1}, H_{n-1} \\ H_{n-1}, -H_{n-1} \end{pmatrix}, H_1 = \begin{pmatrix} 1, 1 \\ 1, -1 \end{pmatrix}$$

и получаем $\vec{\hat{f}} = H_n * (-1)^{\vec{f}}$ - Коэффиценты Адамара-Уолша.

```
TransFormHad[vec_, 1] = {{1, 1}, {1, -1}}.vec;
TransFormHad[vec_, n_] := TransFormHad[vec, n] =
      Module[{tmp1, tmp2, tmp3, tmp4, x},
        If[Length[vec] != 2^n,
          Return[0, Module]
          ];
        tmp1 = TransFormHad[Take[vec, 2^(n - 1)], n - 1];
        tmp2 = TransFormHad[Take[vec, -2^(n - 1)], n - 1];
        tmp3 = tmp1 + tmp2;
        tmp4 = tmp1 - tmp2;
        Join[tmp3, tmp4]
        ];

ClearAll[HadamardCoefficients];
HadamardCoefficients[vec_, n_] :=
    Module[{tmp},
      tmp = 1 - 2vec;
      TransFormHad[tmp, n]
      ];
```

Коэффициенты Фурье получаем из коэффициентов Адамара-Уолша:

$\tilde{f}(\vec{0}) = |f| = \frac{(2^n - \hat{f}(\vec{0}))}{2}$

А для остальных

$\tilde{f}(\vec{\alpha}) = -\frac{\hat{f}(\vec{\alpha}))}{2}$

```
FourierCoefficients[vec_, n_] := Module[{tmp}, tmp = 1 - 2 vec;
      tmp = TransFormHad[tmp, n];
      tmp = tmp/(-2);
      tmp[[1]] = Count[vec, 1];
      tmp
      ];
```

### 2.6 получение таблицы истинности по спектральным коэффициентам

Я принимал за спектральные коэффициенты - коэффициенты Фурье.

Таким образом обращаем получение Фурье из А-У из предыдущего пункта. Применяем преобразование А-У и делим все на $2^n$>, потому что $H_n^{-1} = 2^{-n} * H_n$

Так как мы получили $(-1)^{\vec{f}}$ или же $1 - 2\vec{f}$

Обращаем эту операцию.

```
SpectrToTruth[vec_, n_] :=
    Module[{tmp},
      tmp = -2 vec;
      tmp[[1]] = 2^n + tmp[[1]];
      tmp = TransFormHad[tmp, n]/2^n;
      (1 - tmp)/2
      ];
```

# 3 Разработка способов и реализация средствами САВ “Mathematica” инструментов исследования булевых вектор-функций

## 3.1 Построение вектор-функций для заданных размерностей ходных и выходных векторов

### 3.1.1/2 из разрядных функций/ (псевдо)случайным образом

Та же самая проблема про задание бф как многочлена из пункта 2.1

Поэтому я просто сделал через таблицу истинности вектор функции

```
VectorFunctionR[fs_, n_, m_] := Function[u, Table[fs[[i, FromDigits[u, 2] + 1]], {i, m}]];
```

## 3.2 Исследование разрядных функций вектор-функции

### 3.2.1 Получение разрядных функций вектор-функции, заданной таблично, где входные и выходные вектора упакованы в (целые неотрицательные) числа

Просто разворачиваем таблицу

```
BaseFunctions[vf_, n_, m_] :=
    Module[{tmp, res, i, j},
      res = Table[Table[0, {2^n}], {m}];
      For[i = 0, i < 2^n, i++,
        tmp = IntegerDigits[vf[[i + 1]], 2, m];
        For[j = 0, j < m, j++,
```

```
    res[[j + 1, i + 1]] = tmp[[j + 1]];
    ];
  ];
res
];
```

### 3.2.2 Получение следующих характеристик разрядных функций

- Вес

  Число 1 в таблице истинности

  ```
  Weight[f_] := Count[f, 1];
  ```

- Число мономов для многочлена Жегалкина каждой функции

  Число 1 в векторе АНФ

  ```
  MonomsCountZheg[f_, n_] :=
  Module[{tmp},
    tmp = TransFormVec[f, n];
    Count[tmp, 1]
    ];
  ```

- Число мономов во всех многочленах Жегалкина разрядных функциях

  Число 1 в Побитовом Или векоторов АНФ всех разрядных функций

  ```
  MonomsAll[fs_, n_, m_] :=
  Module[{tmp, i},
    tmp = fs[[1]];
    For[i = 1, i < m, i++,
      tmp = BitOr[tmp, fs[[i + 1]]];
      ];
    Count[tmp, 1]
    ];
  ```

- Список линейных аналогов и соответствующих вероятностей совпадения разрядной функции с линейной (аффинной)

  Считаем коэффициенты А-У. Из них получаем вероятности(не обязательный шаг для сортировки)

  $P(f(\vec{x}) = \vec{\alpha} * \vec{x}) = \frac{1}{2} + \frac{\hat{f}(\vec{\alpha})}{2^{n+1}}$

  Итерируемся по всем полученным вероятностям и ищем максимальную(минимальную). Возвращаем все линейные(афинные) аналоги с данной вероятностью.

  ```
  LinAnalog[f_, n_] :=
    Module[{had, probs, m, res, i, fin},
        had = HadamardCoefficients[f, n];
      probs = 1/2 + had/2^(n + 1);
  ```

```
 m = 0;
res = {};
For[i = 0, i < 2^n, i++,
    If[probs[[i + 1]] > m,
        m = probs[[i + 1]];
        res = {i};,
        If[probs[[i + 1]] == m,
            res = Append[res, i];
            ];
        ];
    ];
fin = Table[IntegerDigits[res[[i]], 2, n], {i, Length[res]}];
Append[fin, m]
];

AffineAnalog[f_, n_] :=
 Module[{had, probs, m, res, i, fin},
    had = HadamardCoefficients[f, n];
  probs = 1/2 - had/2^(n + 1);
  m = 0;
  res = {};
  For[i = 0, i < 2^n, i++,
      If[probs[[i + 1]] > m,
          m = probs[[i + 1]];
          res = {i};,
          If[probs[[i + 1]] == m,
              res = Append[res, i];
              ];
          ];
      ];
  fin = Table[IntegerDigits[res[[i]], 2, n], {i, Length[res]}];
  Append[fin, m]
  ];
```

- Действительный многочлен, его степень и число мономов, а также частные производные в точке (0.5,…,0.5) по каждой переменной
  - Действительный многочлен мы уже считали в пункте 2.4
  - Вероятность равенству 1 - вычисляем многочлен в точке по его коэффицентам `Mathematica` `GetPr[f_, n_, p_] := Module[{res, tmp, tmp1111, mon, vec, i}, vec = TransFormPoly[f, n]; tmp = Table[p, {a, 1, n}]; res = 0; For[i = 0, i < 2^n, i++, If[vec[[i + 1]] == 0, Continue[];]; mon = 1; tmp1111 = IntegerDigits[i, 2, n]; For[j = 0, j < n, j++, If[tmp1111[[j + 1]] == 1,`

```
    mon *= tmp[[j + 1]];                ];           ];
    res += vec[[i + 1]]*mon;            ];       res       ];
```

- Степень - в векторе коэффициентов ищем индекс, вес бинарного представления которого максимален
- Число мономов - считаем ненулевые элементы в векторе коэффициентов
- Частные производные - то же что и вероятность но только убираем те коэффиценты в которых на $i$-й позиции бинарного представления - $0$

## 3.3 Получение коэффициентов Адамара-Уолша вектор-функции и соответственно списка линейных комбинаций входных и выходных переменных, вероятность равенства нулю которых максимальна (минимальна).

Анализируем уже Тетта функцию:

$$\Theta_F(\vec{x}, \vec{y}) = \begin{cases} 1 & \text{Если } \vec{F}(\vec{x}) = \vec{y} \\ 0 & \text{Иначе} \end{cases}$$

Так как

$P(\vec{a} * \vec{x} = \vec{b} * \vec{F}(\vec{x})) = 1/2 + \tilde{\Theta_F}(\vec{a}, \vec{b})/2^{n+1}$

С помощью коэффицентов Фурье ищем наиболее вероятную комбинацию.

Минимальная из таких вероятностей позволяет найти комбинации вида

$\vec{a} * \vec{x} = \vec{b} * \vec{F}(\vec{x}) + 1$

```
VectorToLinearZero[F_, n_, m_] :=
    Module[{fur, probs, mprob, res, i, fin, tmp},
      fur = FourierCoefficients[F, n + m];
      probs = 1/2 + fur/2^(n + 1);
      mprob = 0;
      res = {};
      For[i = 1, i < 2^(m + n), i++,
        If[probs[[i + 1]] > mprob,
          mprob = probs[[i + 1]];
          res = {i};,
          If[probs[[i + 1]] == mprob,
            res = Append[res, i];
            ];
          ];
        ];

      fin = Table[
```

```
            tmp = IntegerDigits[res[[i]], 2, n + m];
            {Take[tmp, n], Take[tmp, -m]},
            {i, Length[res]}
            ];
        Append[fin, mprob]
        ];

VectorToLinearOne[F_, n_, m_] :=
    Module[{fur, probs, mprob, res, i, fin, tmp},
      fur = FourierCoefficients[F, n + m];
      probs = 1/2 - fur/2^(n + 1);
      mprob = 0;
      res = {};
      For[i = 0, i < 2^(m + n), i++,
        If[probs[[i + 1]] > mprob,
            mprob = probs[[i + 1]];
            res = {i};,
            If[probs[[i + 1]] == mprob,
                res = Append[res, i];
                ];
            ];
        ];

      fin = Table[
          tmp = IntegerDigits[res[[i]], 2, n + m];
          {Take[tmp, n], Take[tmp, -m]},
          {i, Length[res]}
          ];
      Append[fin, mprob]
      ];
```