1. Код Рида-Маллера $\boldsymbol{RM(n,r)}$.

$$RM(n,r) = \{f \in \mathbb{F}_2[x_1, x_2, \dots, x_n] : \deg f \leq r\}$$

$d = 2^{n-r}$ – кодовое расстояние

$t = 2^{n-r-1} - 1$ – число исправляемых ошибок

2. Кодирование:

применение преобразования Мебиуса.

2.1) Сложность последовательного варианта: (T – число операций, M – память)

$$T_1(n) = \frac{2^n}{2} + 2T_1(n-1) = \frac{2^n}{2} + 2\left(\frac{2^n}{4} + 2T_1(n-2)\right) = \frac{k2^n}{2} + 2^k T_k(n-k) = \frac{n2^n}{2} = O(n2^n)$$

$M_1(n) = O(1)$

2.2) Сложность параллельного варианта (p вычислительных единиц)

Если $p \leq 2^{n-1}$ то

$$T_p(n) = \sum_{i=1}^{n} \frac{2^n}{2p} = \frac{n2^n}{2p} = O\left(\frac{n2^n}{p}\right)$$

Если $p \geq 2^{n-1}$ то

$$T_p(n) = T_{2^{n-1}}(n) = \frac{n2^n}{2 * 2^{n-1}} = n = O(n)$$

Для любого p:

$M_p(n) = O(1)$

3. Декодирование:

алгоритм Рида.

3.1) Для всех мономов степени k, при $k > 1$

Число мономов степени k: C_n^k. Число граней для проверки монома 2^{n-k}. В одной грани 2^k бит, которые нужно проксорить. Затем нужно найти, какое значение встречается чаще среди сумм граней.

Сложность последовательного варианта:

$T_{k,1}(n) = C_n^k(2^{n-k}2^k + 2^{n-k})$

$M_{k,1}(n) = 2^{n-k}$ (2^{n-k} сумм граней для каждого монома, при последовательном вычислении в один момент времени хранятся суммы только для одного монома)

Сложность параллельного варианта (для p процессоров):

В нашей реализации алгоритма, вычисления для различных мономов и граней проводятся параллельно, а ксоры по граням и определение наиболее частого значения вычисляется последовательно. Отсюда получаем следующие оценки для числа операций и памяти:

$$T_{k,p}(n) = \begin{cases} \dfrac{T_{k,1}(n)}{p}, & p \le C_n^k \\ 2^{n-k} + \dfrac{C_n^k 2^n}{p}, & p \in [C_n^k, C_n^k 2^{n-k}] \\ 2^{n-k} + 2^k, & p \ge C_n^k 2^{n-k} \end{cases}$$

$$M_{k,p}(n) = \begin{cases} p2^{n-k}, & p \le C_n^k \\ C_n^k 2^{n-k}, & p \ge C_n^k \end{cases}$$

3.2) Для всех мономов степени k, при $k \le 1$

Для $k \le 1$ мы применяем алгоритм Лицына-Шеховцева, описание которого приведено в разделе 4.

Сложность шага алгоритма Лицына-Шеховцева для набора длины 2^n: $2^{n-1} + 2^n$ (вычисление одного шага преобразования Уолша и суммирование модулей вектора)

Отсюда, сложность последовательного варианта:

$T_{1,1}(n) = \sum_{i=1}^{n} 2^{n-i} + 2^{n+1-i} \le 2^n \sum_{i=1}^{\infty} 2^{-i} + 2^{1-i} = 3 * 2^n = O(2^n)$

$M_{1,1}(n) = O(2^n)$ (необходимо скопировать вектор функции)

Сложность параллельного варианта:

Так как в нашей реализации суммирование модулей осуществляется последовательно, распараллеливание никак не влияет на второе слагаемое в выражении для сложности шага алгоритма. Соответственно асимптотическая сложность параллельной версии совпадает с последовательной:

$T_{1,p}(n) = O(2^n)$

$M_{1,p}(n) = O(2^n)$

3.3) Общая сложность

Сложность последовательного варианта:

$T_1(n) = \left(\sum_{k=2}^{r} C_n^k (2^{n-k}2^k + 2^{n-k}) + O(2^n) + O(rn2^n)\right) = O\left(\sum_{k=0}^{r} n^k 2^n\right) = O(n^r 2^n)$ (выполняем исключение всех мономов до 2-ой степени, затем применяем алгоритм Лицына-Шеховцева, для исключения мономов применяем преобразование Мёбиуса, использовалась оценка $C_n^k = O(n^k)$)

$M_1(n) = O\left(\max_k 2^{n-k}\right) = O(2^{n-2}) = O(2^n)$

Сложность параллельного варианта:

$T_p(n) = \sum_{k=2}^{r} T_{k,p}(n) + O(2^n) + O\left(\frac{rn2^n}{p}\right) + O\left(\frac{r2^n}{p}\right)$ $= \sum_{k=2}^{r} T_{k,p}(n) + O(2^n) + O\left(\frac{rn2^n}{p}\right)$ ($\frac{n2^n}{p}$ – сложность преобразования Мёбиуса необходимого для исключения мономов, $\frac{2^n}{p}$ – сложность параллельного ксора)

Так как в сумме $\sum_{k=2}^{r} T_{k,p}(n)$ каждое из слагаемых описывается довольно сложным выражением, затруднительно оценить значение этой суммы для произвольного p, поэтому рассмотрим пару частных случаев.

При $p \le C_n^2$ справедливо выражение:

$$T_p(n) = \frac{1}{p}\sum_{k=2}^{r} T_{k,1}(n) + O(2^n) + O\left(\frac{rn2^n}{p}\right) = \frac{1}{p}\sum_{k=2}^{r} C_n^k(2^{n-k} + 2^k) + O(2^n) + O\left(\frac{rn2^n}{p}\right) = O\left(\frac{n^r 2^n}{p} + 2^n\right)$$

А при $p^* = \max_k(2^{n-k}\, C_n^k)$ достигаеится максимально возможное распараллеливание:

$$T_{p^*}(n) = \sum_{k=2}^{r} 2^{n-k} + 2^k + O(2^n) + O(rn) = O(2^n)$$

Сравнивая это выражения со сложностью последовательного алгоритма можно увидеть, что ускорение равно n^r.

Требуемая память:

$$M_p(n) = \begin{cases} O(p2^{n-1}), & p \le C_n^k \\ O(\max_k(2^{n-k}\, C_n^k)), & p \ge C_n^k \end{cases}$$

4. Об алгоритме Лицына-Шеховцева

Алгоритм:

Вход: (w, n, r)

1) $i = 2^n, j = 1.$
2) S = матрица размерности $2 \times \left(\frac{i}{2}\right)$, $\forall k = 1 \ldots \frac{i}{2} \Rightarrow S[1,k] = w[2k], S[2,k] = w[2k+1]$
3) $S = H_2 S$
4) s – вектор-столбец высоты 2. $s[1] = \sum_{k=1}^{\frac{i}{2}} |S[1,k]|$, $s[2] = \sum_{k=1}^{\frac{i}{2}} |S[2,k]|$
5) Если $s[1] < s[2]$, коэффициент при x_j равен 1, иначе 0.
6) Если $j < r$
   a. Если $s[1] < s[2]$, то $\forall k = 1 \ldots \frac{i}{2} \Rightarrow S[1,k] = S[2,k]$
   b. $w = S[1,:], j = j+1, i = \frac{i}{2}$,
   c. goto 2
7) Если $w[1] < 0$, свободный член равен 1, иначе 0.

Рассмотрим пример декодирования этим методом

Рассмотрим пример декодирования $x_1 + x_3 + 1 \in RM(4,1), d = 8, t = 3$, в вектор значений которой внесено 3 ошибки. В векторе они выделены

x_1	x_2	x_3	x_4	$f(x)$	$f(x)+e$	$\widehat{f(x)+e}$	(1)	(2)	(3)	(4)	(5)
0	0	0	0	1	1	-1	-2				
0	0	0	1	1	1	-1	0				
0	0	1	0	0	0	1	0				
0	0	1	1	0	**1**	-1	0				
0	1	0	0	1	1	-1	0				
0	1	0	1	1	1	-1	0				
0	1	1	0	0	0	1	0				
0	1	1	1	0	0	1	0				
1	0	0	0	0	**1**	-1	0	-2	2		
1	0	0	1	0	0	1	-2	-4	-4		
1	0	1	0	1	1	-1	2	4	-6	-10	-10
1	0	1	1	1	**0**	1	-2	0	-4	-2	
1	1	0	0	0	0	1	-2	2			

1	1	0	1	0	0	1	-2	0			
1	1	1	0	1	1	-1	2	0			
1	1	1	1	1	1	-1	2	-4			

(1) Сумма модулей сверху – 2, снизу – 14 => $x_1 = 1$, сносим нижнюю половину.
(2) Сумма модулей сверху – 10, снизу – 6 => $x_2 = 0$, сносим верхнюю половину.
(3) Сумма модулей сверху – 6, снизу – 10 => $x_3 = 1$, сносим нижнюю половину.
(4) Сумма модулей сверху – 10, снизу – 2 => $x_4 = 0$, сносим верхнюю половину.

-10 < 0 => свободный член равен 1.

Основная идея этого метода заключается в том, чтобы найти такие пары наборов, на которых, в случае вхождения определенного монома в АНФ кодового слова, были разные значения, а в случае отсутствия – одинаковые.

Попытки обобщить этот подход на случай кода произвольного порядка не увенчались успехом.

Например, в частном случае разделимости по переменным проверяемого монома (пусть этот моном имеет вид $x_1x_2 \dots x_r$, а сама функция имеет особый вид $f(x_1, \dots, x_n) = x_1x_2 \dots x_r + g(x_{r+1}, \dots, x_n)$ наличие монома $x_1x_2 \dots x_r$ в АНФ можно проверить аналогичным способом: если он не входит в АНФ, то $f(1, \dots 1, x_{r+1}, \dots, x_n) = f(1, \dots 0, x_{r+1}, \dots, x_n)$, иначе $f(1, \dots 1, x_{r+1}, \dots, x_n) = \overline{f(1, \dots 0, x_{r+1}, \dots, x_n)}$

Если функция имеет чуть более общий вид с разделением по переменной $f(x_1, \dots, x_n) = h(x_1, x_2, \dots, x_r) + g(x_{r+1}, \dots, x_n) + c$, то отсутствие переменной x_1 в АНФ можно проверить так

$f(1,0, \dots 0, x_{r+1}, \dots, x_n) = f(0, \dots 0, x_{r+1}, \dots, x_n)$. Т.е. если можно отделить через сумму подфункцию от более, чем $n - r$ переменной, то декодировать АНФ другой подфункции можно как в алгоритме выше.

Справедливо общее разложение по r переменным

$$f(x_1, \dots, x_n) = \bigoplus_{\sigma \in F_2^r} a_\sigma(x_{r+1}, \dots, x_n)(x_1x_2 \dots x_r)^\sigma = \bigoplus_{\sigma \in F_2^r} a_\sigma(x_{r+1}, \dots, x_n)x_1^{\sigma[1]}x_2^{\sigma[2]} \dots x_r^{\sigma[r]}$$

При фиксировании переменных $x_{r+1}, \dots, x_n$ получаются различные подфункции функции f от переменных $x_1x_2 \dots x_r$, при этом, так как известно, что во всей функции коэффициент перед мономом $x_1x_2 \dots x_r$ должен быть константой, то у всех этих подфункций коэффициент перед $x_1x_2 \dots x_r$ должен быть одинаковым и равен этой же константе. Применение одного шага преобразования Мебиуса к этим подфункциям с целью определить коэффициент при $x_1x_2 \dots x_r$ в каждой из них с последующим применением мажоритарной логики дает алгоритм Рида.