Дмитрий Герасимов (4538)
Теория кодирования **Задание 4** 16 декабря 2013 г.

Задача 1

Рассмотим произвольный линейный код $\mathcal{C}$, для которого хотим найти минимальное расстояние d, пусть для него построена решетка (будем обозначать ее как граф $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$). Обозначим за $\texttt{pasts}(v)$ — множество прошлых, заканчивающихся в вершине v, оно пересчитывается как $\texttt{pasts}(v) = \bigcup\limits_{v'v \in \mathbb{E}} \{p'c \mid p' \in \texttt{pasts}(v')\}$ (где c — символ, написанный на соответствующем ребре. Найдем минимальное расстояние методом динамического программирования: будем поддерживать в каждом узле решетки два числа a_v и b_v — два наименьших веса различных прошлых, лежащих в $\texttt{pasts}(v)$.

В начале работы алгоритма на нулевом слое в его единственной вершине присвоим $a_v = 0$ и $b_v = \infty$.

Пусть до какого-то слоя i все такие значения подсчитаны, рассматриваем некий узел v слоя i. Нам нужно найти два минимальных значения из множества (а точнее, мультимножества, так как у двух различных прошлых может быть один вес) весов всех путей в $\texttt{pasts}(v)$: запишем его как

$$W(v) = \{w(p) \mid p \in \texttt{pasts(v)}\} = \bigcup_{v'v \in \mathbb{E}} \{w(p'c) \mid p' \in \texttt{pasts}(v')\} = \bigcup_{v'v \in \mathbb{E}} \{w(p')+c \mid p' \in \texttt{pasts}(v')\}$$

$$W(v) = \bigcup_{v'v \in \mathbb{E}} \{w' + c \mid w' \in W(v')\}$$

Пересчет минимума из всего W даже пользуясь уже подсчитанными весами $W(v')$ занял бы слишком много времени. Однако воспользуемся тем, что у нас уже подсчитаны $a_{v'}$ и $b_{v'}$, тогда утверждается, что можно взять минимумы из множества $\bigcup\limits_{v'v \in \mathbb{E}} \{a_{v'} + e, b_{v'} + e\}$. Докажем от противного: пусть существуют более оптимальные c, d, то есть:

- либо $c < a_v$, что будет означать, что существует прошлое p', приходящее в некоторую вершину v' слоя $i - 1$, из которой есть ребро в нашу вершину v, тогда $c = w(p) = w(p') + e$, причем это $w(p')$ равно $a_{v'}$, иначе c можно было бы еще уменьшить. Но значение $a_{v'} + e$ по построению есть в множестве $W(v)$, то есть минимальное значение не могло быть больше c, получили противоречие.
- либо $c = a_v$ и $d < b_v$. Аналогично, это будет значить, что существует некоторое прошлое p', приходящее в вершину v' слоя $i - 1$, из которой есть ребро в нашу вершину v, такое что $d = w(p) = w(p') + e$. Есть два случая:
  - вершина v' — та же, через которую проходит кратчайший путь в v. Но тогда в этой вершине мы сможем улучшить $b_{v'}$, то есть получили противоречие аналогично предыдущему пункту.
  - вершина v' — не та, через которую проходит кратчайший путь в v. Но тогда в этой вершине мы сможем улучшить $a_{v'}$, то есть получим противоречие по аргументу аналогично предыдущему пункту.

Таким образом, предложенный алгоритм действительно считает два минимальных пути. В конце его работы в единственной вершине v последнего слоя значение a_v будет равно 0, так как в линейном коде всегда существует нулевое слово, а b_v будет равно весу минимального ненулевого пути, то есть это и будет минимальным расстоянием.

Алгоритм требует линейное время относительно суммарного количества узлов в решетке (так как количество ребер в этой решетке не превышает удвоенного количества узлов) и линейной относительно максимальной сложности решетки памяти (так как достаточно хранить значения a и b только для текущего и предыдущего слоев).

Задача 3

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Принята последовательность $y = 01111$.

Вычислим функцию правдоподобия $p(y \mid c)$, и с помощью нее — апостериорные вероятности в предположении, что входные вероятности кодовых слов одинаковы.

Информационные символы u_1, u_2	Кодовое слово	$p(y \mid c)$	$p(c \mid y)$
00	00000	$(1-p_0)^1 p_0^4 = 0.00009$	0.0012
01	10111	$(1-p_0)^3 p_0^2 = 0.00729$	0.0988
10	11010	$(1-p_0)^2 p_0^3 = 0.00081$	0.011
11	01101	$(1-p_0)^4 p_0^1 = 0.06561$	0.889

Символ	$\ln \frac{p(1\mid y)}{p(0\mid y)}$
u_1	2.197225
u_2	4.394449
c_1	−2.093235
c_2	2.197225
c_3	4.394449
c_4	−2.093235
c_5	4.394449

Жесткое решение будет принято в пользу 01101.

Задача 4

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

1. Построим решетку по порождающей матрице, для этого сначала приведем ее в МСФ:

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

2. Синдромная решетка строится непосредственно по проверочной матрице H.

Задача 5

В результате расчетов был получен вектор $\gamma = (-2.093, 2.197, 4.394, -2.093, 4.394)$, что совпадает с результатом, полученным в задаче 3, жесткое решение будет принято в пользу кодового слова 01101.

Задача 6

Далее обозначим за S суммарное количество вершин в решетке, а за M — ее максимальную сложность.

- Переборное декодирование: в его ходе происходит перебор по всем возможным путям в решетке, в худшем случае таких путей может быть порядка 2^n. Для вычисления веса каждого пути нужно n операций, также будет порядка 2^n операций сравнения для выбора минимального. Таким образом, время работы алгоритма — порядка $O(n2^n)$.

  Алгоритму необходимо $O(n)$ памяти для хранения состояния перебора на каждом слое сети.

- Декодирование алгоритмом Витерби: на каждом слое необходимо сделать столько операций, сколько в него идет ребер, то есть в сумме нужно сделать количество операций, равное сумме количества ребер и количества вершин в решетке. Так как из каждой вершины выходит не более двух ребер, получаем линейную сложность отнисительно $O(S)$. Известно, что максимальная сложность решетки M ограничена величиной $\min(2^k, 2^{n-k})$, тогда суммарное количество вершин можно ограничить величиной $n\min(2^k, 2^{n-k})$, и получаем асимптотику $O(n\min(2^k, 2^{n-k}))$.

  По реализации работы с памятью есть два решения — можно заметить, что хранить минимальные значения для вычисления ответа достаточно только для текущего и предыдущего слоев. Но тогда для предыдущего слоя придется хранить последовательности ребер, на которых они достигаются. Этому алгоритму гарантированно нужно $O(nM)$ памяти, что можно оценить как $O(n\min(2^k, 2^{n-k}))$.

  Восстановить ответ можно, двигаясь по слоям с конца, то есть надо хранить минимальные значения для всех вершин, необходимо $O(S)$ памяти, что также можно оценить как $O(n\min(2^k, 2^{n-k})$.

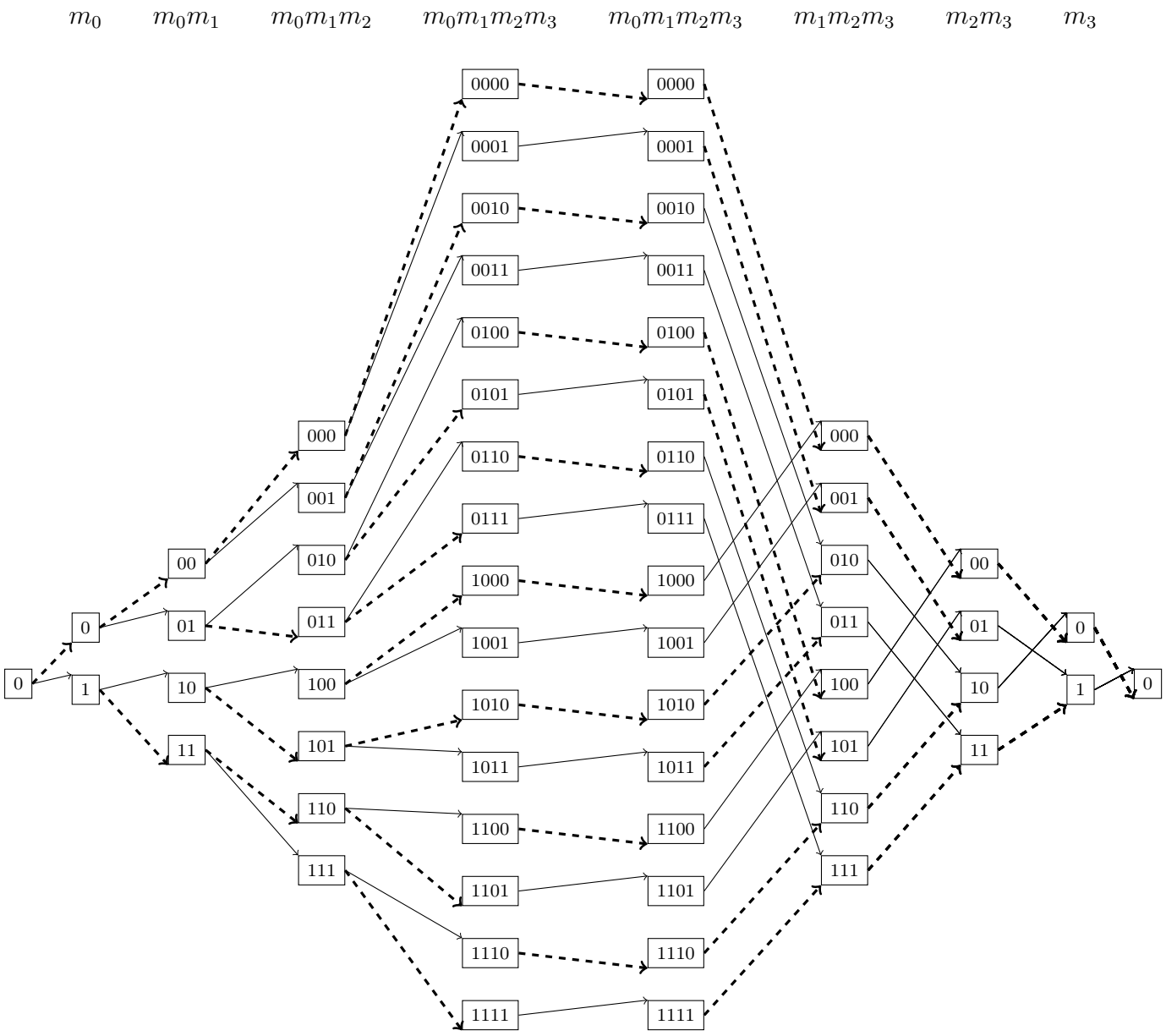

Рис. 1: Решетка для кода к заданию 4. Пунктиром — ребра, на которых написано 0, целыми линиями — на которых написано 1.

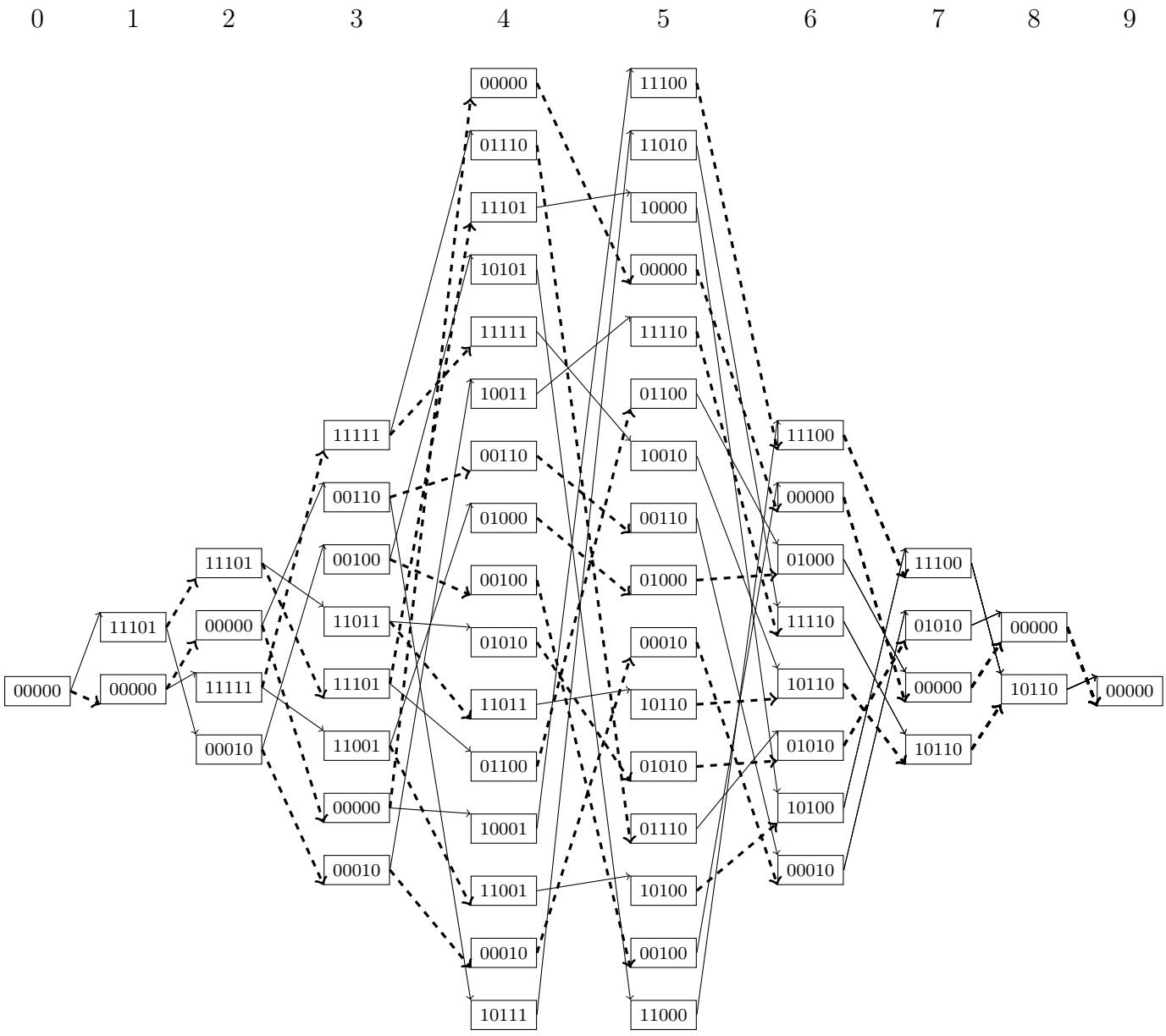

Рис. 2: Синдромная решетка для кода к заданию 4. Пунктиром — ребра, на которых написано 0, целыми линиями — на которых написано 1

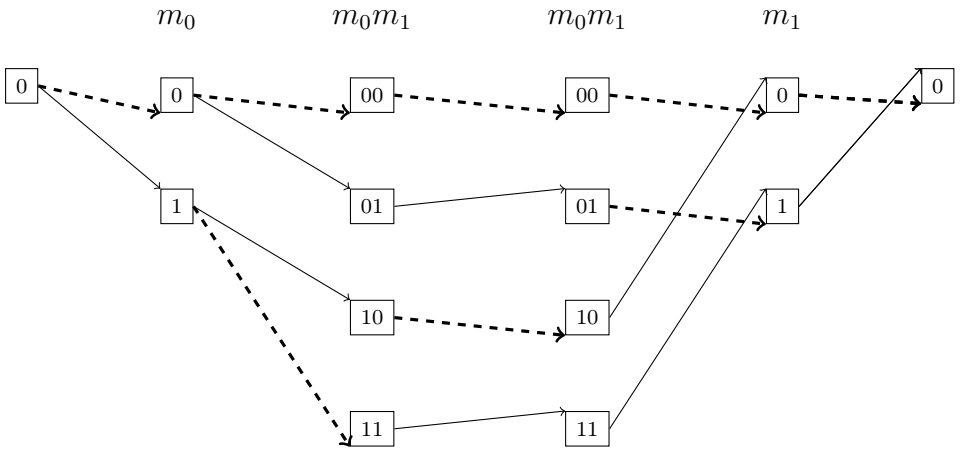

Рис. 3: Решетка для кода к заданию 5. Пунктиром — ребра, на которых написано 0, целыми линиями — на которых написано 1

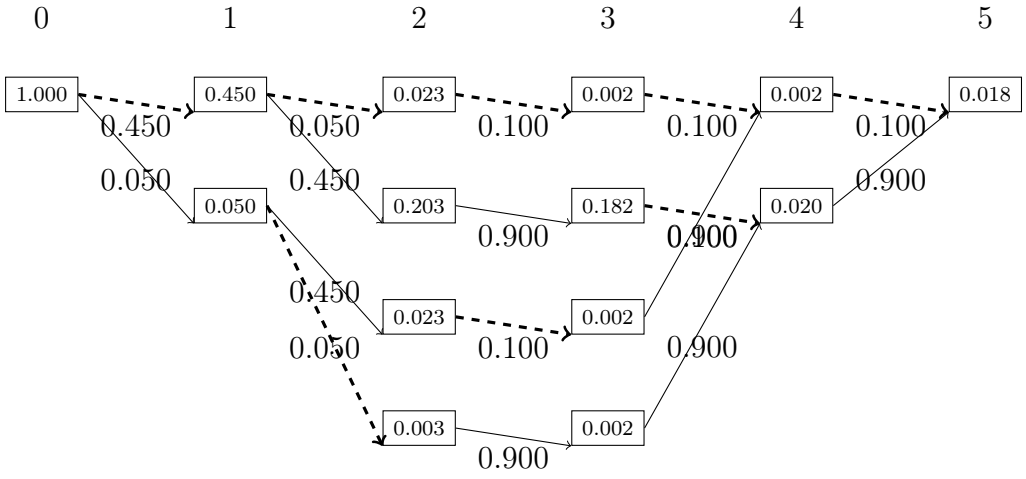

Рис. 4: Решетка для кода к заданию 5, показаны значения α и γ.

- Декодирование алгоритмом БКДР: аналогично алгоритму Витерби на каждом слое делается количество операций, равное количеству входящих (или исходящих, в зависимости от того, что вычисляем) в слой ребер, для принятия жесткого решения нужно $O(n)$ операций, то есть суммарно необходимо $O(n \min(2^k, 2^{n-k}))$ операций.

  Для вычисления значений σ надо знать одновременно и α, и β, и γ , поэтому придется хранить в ходе алгоритма все $O(n \min(2^k, 2^{n-k}))$ значений, что является оценкой памяти, нужной алгоритму.

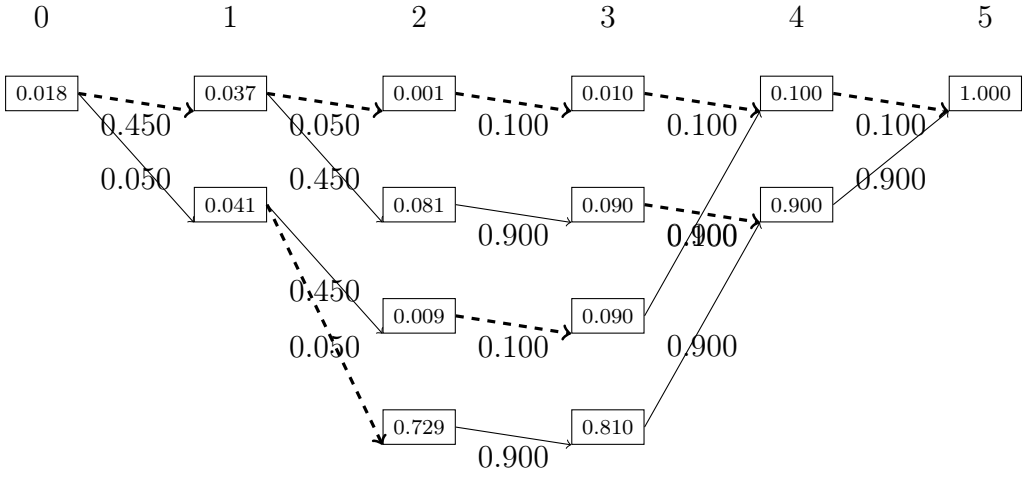

Рис. 5: Решетка для кода к заданию 5, показаны значения β и γ.

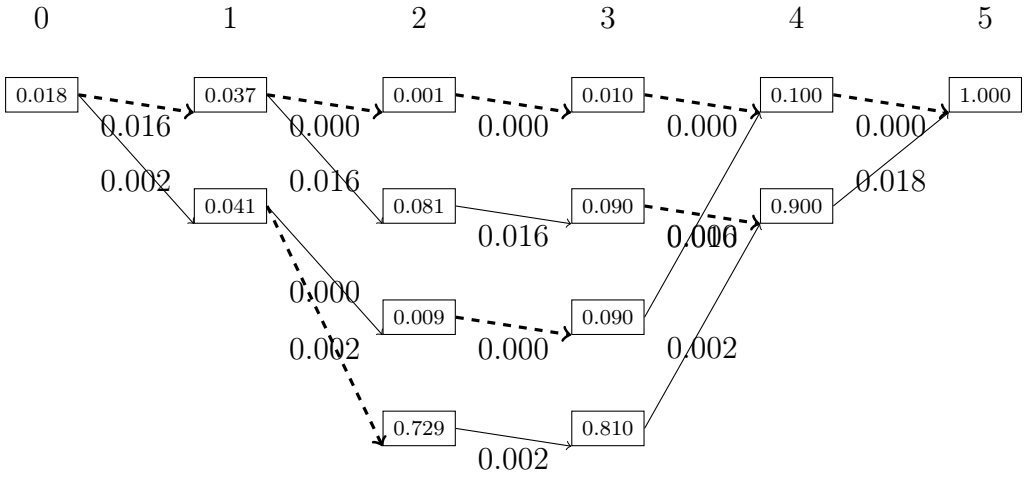

Рис. 6: Решетка для кода к заданию 5, показаны значения β и σ.